
IzPack documentation

<http://www.izforge.com/izpack/>

Edition of January 11, 2005

Copyright © 2001-2004

- Julien PONGE <julien@izforge.com>
- Elmar GROM <elmar@grom.net>
- Tino SCHWARZE <tino.schwarze@informatik.tu-chemnitz.de>
- Klaus BARTZ <klaus.bartz@coi.de>

This documentation is licensed under the terms of the *Commons Creative Attribution-NonCommercial-ShareAlike* license version 1.0. You can use it and modify it under certain conditions. See page [108](#) for the legal terms.

Contents

1	Getting started	10
1.1	Overview	10
1.2	First Compilation	10
1.3	The IzPACK Architecture	11
1.3.1	The Compilation System	11
1.3.2	How an Installer Works	12
1.3.3	The Different Kinds of Installers	13
2	Writing Installation XML Files	14
2.1	What You Need	14
2.1.1	Your editor	14
2.1.2	Writing XML	14
2.2	Variable Substitution	15
2.2.1	The Built-In Variables	16
2.2.2	Environment Variables	16
2.2.3	Parse Types	17
2.3	The IzPACK Elements	17
2.3.1	The Root Element <code><installation></code>	17
2.3.2	The Information Element <code><info></code>	17
2.3.3	The Variables Element <code><variables></code>	19
2.3.4	The GUI Preferences Element <code><guiprefs></code>	19
2.3.5	The Localization Element <code><locale></code>	21
2.3.6	The Resources Element <code><resources></code>	22
2.3.7	The Panels Element <code><panels></code>	23
2.3.8	The Packs Element <code><packs></code>	24
	Internationalization of the PacksPanel	24
	<code><description></code> - pack description	25
	<code><depends></code> - pack dependencies	25
	<code><os></code> - OS restrictions	25
	<code><updatecheck></code>	25
	<code><file></code> - add files or directories	26

	<code><additionaldata></code>	26
	<code><singlefile></code> - add a single file	26
	<code><fileset></code> : add a fileset	27
	<code><parsable></code> - parse a file after installation	28
	<code><executable></code> - mark file executable or execute it	28
	<code><os></code> - make a file OS-dependent	29
2.3.9	The Native Element <code><native></code>	30
	<code><os></code> - make a library OS-dependent	31
2.3.10	The Jar Merging Element <code><jar></code>	31
2.4	The Available Panels	31
2.4.1	HelloPanel	31
2.4.2	InfoPanel and HTMLInfoPanel	32
2.4.3	LicencePanel and HTMLLicencePanel	32
2.4.4	PacksPanel	32
2.4.5	ImgPacksPanel	32
2.4.6	TargetPanel	32
2.4.7	InstallPanel	33
2.4.8	XInfoPanel	33
2.4.9	FinishPanel	33
2.4.10	SimpleFinishPanel	33
2.4.11	ShortcutPanel	34
2.4.12	UserInputPanel	34
2.4.13	CompilePanel	34
2.4.14	ProcessPanel	35
2.4.15	JDKPathPanel	37
3	Advanced Features	39
3.1	Ant Integration	39
3.2	System properties as variable	40
3.3	Automated Installers	40
3.4	Picture on the Language Selection Dialog	41
3.5	Picture in the installer	41
3.6	Web Installers	41
3.7	More Internationalization	42
3.7.1	Special resources	42
3.7.2	Packs	42
4	Desktop Shortcuts	44
4.1	Defining Shortcuts	44
4.1.1	Introduction	44
4.1.2	What to Add to the Installer	45

4.1.3	Why Native Code to do the Job on Windows?	47
4.1.4	The Shortcut Specification	48
4.1.5	Shortcut Attributes	50
	Unix specific shortcut attributes	53
4.1.6	Selective Creation of Shortcuts	54
4.1.7	Summary	55
4.2	Shortcut Tips	55
4.2.1	The Desktop	56
4.2.2	Icons	57
4.2.3	Targets	58
4.2.4	Command Line	60
4.3	Trouble Shooting	61
4.3.1	Problems You Can Solve	61
4.3.2	Problems That Have No Solution (yet)	63
4.3.3	A sample shortcut specification file for Unix	63
5	Creating Your Own Panels	66
5.1	How It Works	66
5.1.1	What You Need	66
5.1.2	What You Have To Do	66
5.2	The IzPanel Class	67
5.2.1	UML Diagram	67
5.2.2	Description	67
6	User Input	69
6.1	The Basic XML Structure	71
6.2	Concepts and XML Elements Common to All Fields	71
6.3	Internationalization	73
6.4	Panel Title	74
6.5	Static Text	75
6.6	Visual Separation	75
6.7	Text Input	76
6.8	Radio Buttons	77
6.9	Combo Box	78
6.10	Check Box	78
6.11	Rule Input	80
6.11.1	Layout and Input Rules	80
6.11.2	Setting Field Content	83
6.11.3	The Output Format	83
6.11.4	Validating the Field Content	84
	NotEmptyValidator	84

	RegularExpressionValidator	84
	Creation Your Own Custom Validator	85
6.11.5	Processing the Field Content	85
6.11.6	Summary Example	85
6.12	Search	86
6.12.1	Specification	86
6.12.2	Example	87
7	Custom Actions	88
7.1	Overview	88
7.2	How It Works	89
7.2.1	Custom Action Types	89
	Custom Actions At Packaging	90
	UML Diagram	90
	Description	90
	Custom Actions At Installing Time	90
	UML Diagram	90
	Description	91
	Custom Actions At Uninstalling Time	92
	UML Diagram	92
	Description	92
7.2.2	Package Path	93
7.2.3	Correlated Stuff	93
	Native Libraries for Uninstallation	93
7.3	What You Have To Do	93
7.3.1	Custom Actions at Packaging (CompilerListener)	94
7.3.2	Custom Actions at Installation Time (InstallerListener)	94
7.3.3	Custom Actions at Uninstallation Time (UninstallerLis- tener)	94
7.4	Example	95
7.5	Ant Actions (InstallerListener and UninstallerListener)	95
7.5.1	The Basic XML Struture	97
	<property>: define a property	98
	<propertyfile>: define properties in a file	98
	<target>: target to call at installation	99
	<uninstall_target>: target to call on uninstallation	99
A	The GNU General Public License	100
B	The Commons Creative Attribution-NonCommercial-ShareAlike License	108

Introduction

Welcome to IzPack !



IzPACK is a tool that will help you to solve your software installation problems. It is a JavaTM based software installer builder that will run on any operating system coming with a *Java Virtual Machine (JVM)* that is compliant with the Sun JVM 1.2 or higher. Its design is very modular and you will be able to choose how **you** want your installer to look and you will also be able to customize it using a very simple *Application Programming Interface (API)*. Although IzPACK is essentially a JavaTM only application (it can run on virtually any operating system), it can interact in a clean way with the underlying operating system. Native code can interact with it on a specific platform without disturbing the operation on incompatible operating systems. For instance, you can develop Unix-specific code that will be silent if run on Windows. To put it in a nutshell, whereas most of the other JavaTM installers force you to go their way, IzPACK will let you go **your way**. Some respectable companies have been

using it in order to produce customized installers for their very specific needs.

"So, if it's so good, how much is it ?" : well, you can get it for free. **BUT** IZPACK is not a *freeware*. It's not *free* as in *"free beer"* but *"free as in free speech"*. So it's neither *freeware* nor *public domain*. It is software covered by the GNU GENERAL PUBLIC LICENSE (GPL). It uses the tactic of *copyleft* : to make it short, you can use it, modify it and redistribute it freely but you must also make your modifications available to everyone whenever you publish a modified version of a *copylefted* software. You have access to the IZPACK source code and you can modify it to make it suit your needs, but if you publish such a modified version, you are forced to publish the modifications you've made. That's a fair exchange of expertise and work. To learn more about the GPL license and the *copyleft* principles, visit <http://www.gnu.org/>.

The Features

IZPACK uses XML files to describe installations. When you make an installer, you have a choice of panels. You can see panels as a kind of plugin that composes the installer. For instance, a panel can choose the installation path, the packs to install, prompt the user for a license agreement and so on. This approach is very modular. You can also create your own panels if you have specific needs. In some cases you even have a choice from multiple panel versions for the same task. You can also choose the order in which panels appear during the installation process. IZPACK can be used in a number of different ways:

- by writing the XML installation file "by hand" and compiling it with the command line compiler
- by invoking the compiler from the great APACHE JAKARTA ANT tool (see <http://jakarta.apache.org/>) as IZPACK can be used as a task for ANT

Here is a brief (and certainly incomplete !) list of the main IZPACK features :

- XML based installation files
- easy internationalization using XML files (10 translations are already available)

- Ant integration, command-line compiler
- easy customization with the panels and a rich API (even an XML parser is included !)
- powerful variable substitution system that you can use to customize scripts and more generally any text-based file
- different kinds of installers (standard, web-based, ...)
- launching of external executables during the installation process and Unix executable flag support (useful for the scripts for instance)
- layout of the installation files in packs (some can be optional)
- native code integration facilities
- jar files nesting support
- ... *more things to discover and create !.*

The Development

I started writing IzPACK in April 2001 and many people have helped me improving it since. I prefer not to mention them here as I would for sure forget some of them, so please check the file named **Thanks.txt** which I try to get as up-to-date as possible in order to mention everyone who helped me. As far as I'm concerned, I'm a french student and I rather see this as a fun activity in my free time where I can learn a lot of great things. The contributors to the project are both individuals and companies. Help can take any form :

- translations
- new features and various fixes
- bug fixes
- writing manuals
- ... anything else you like :-)

The official IZPACK homepage is located at <http://www.izforge.com/izpack/>. The IzPack developer services (mailing-lists, CVS, patches manager, bugs tracker, ...) are generously hosted by BerliOS. The IzPack BerliOS section is located at <http://developer.berlios.de/projects/izpack/>. Feel free to use these services. In particular, there are two mailing-lists:

- `izpack-devel`: used for the IzPack development
- `izpack-users`: general users lounge, great to get some help with IzPack.

3rd party code used in IzPack

IZPACK uses several 3rd party libraries and I would like to mention them in respect for their respective authors work :

- *NanoXML* by Marc DE SCHEEMAECKER : the XML parser used inside IZPACK and released under a *zlib/png*-style license - see <http://nanoxml.sourceforge.net/> -
- *Kunststoff Look and Feel* by Incors Gmbh : a Swing™ Look and Feel that can be used for installers. It **really** looks good and is released under the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) - see <http://www.incors.org/> -
- *Crystal-SVG Icons* : the icons used in IZPACK come from the great work of Everaldo (<http://www.everaldo.com/>) that makes KDE 3.2 look so sweet
- *Some Apache Jakarta classes and libraries* : released under the *Apache License*
- *Metouia Look and Feel* by Taoufik Romdhane : released under the *LGPL license* - see <http://mlf.sf.net/>
- *Liquid Look and Feel* by Miroslav Lazarevic : released under the *LGPL license* - see liquidlnf.sf.net/
- *JGoodies Looks* by Karsten Lentzsch : released under a *BSD-style license* - see <http://looks.dev.java.net/>.

So, now let's dive into understanding how IZPACK works. You'll be surprised to see how powerful and simple it can be :-)

Chapter 1

Getting started

1.1 Overview

To begin with, you should know what IZPACK is organized if you want to use it. Let's go into the directory where you have installed IZPACK on your machine. There are 3 text files and a set of directories. The most important for the moment are `bin/` `doc/` `sample/`. If you are reading this, you already know that `doc` contains this documentation :-)

So let's go into `bin/`. The `icons/` directory contains some directories for your system, in case you would like an icon to launch a component of IZPACK . But the most important things you can see in `bin` are the `compile` scripts (in both Unix* and Windows formats). `compile` is used to compile a ready-to-go XML installation file from a command-line context or from an external tool.

Note : these scripts can be launched from anywhere on your system as the installer has customized these scripts so that they can inform IZPACK of where it is located.

1.2 First Compilation

Now you probably can't wait to build your first installer. So go on open a command-line shell and navigate to `sample/`. The following should work on both Unix* and Windows systems. For the latter, just change the path separator (slash '/') to a backslash. So type (\$ is your shell prompt !) :

```
$ ../bin/compile install.xml -b . -o install.jar -k standard
```

```
(installer generation text output here)
$ java -jar install.jar
```

There you are! The first command has produced the installer and the second one did launch it.

1.3 The IzPack Architecture

Now that you have packaged your first installer, it's time for you to understand how the whole thing works.

1.3.1 The Compilation System

The compilation system (see figure 1.1) is quite modular. Indeed, you can use the compiler in 2 ways :

- from a command-line
- from Jakarta Ant

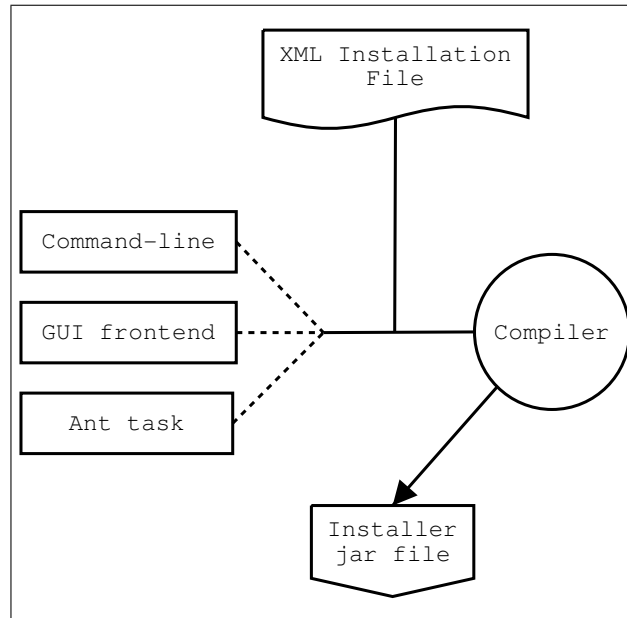
The compiler takes as its input an XML installation file that describes (at a relatively high-level) the installation. This file contains detailed information such as the application name, the authors, the files to install, the panels to use, which resources to load and much more (see figure 1.2).

The compiler can generate different kinds of installers, but this information is not located inside the XML file as it is not where it should be. On the contrary, this is a compiler parameter.

The compilation options for a command-line installer are the following:

- `-?`: gives a list of the available options.
- `-b`: specifies the base path, ie the one that will be used to resolve the relative paths. If your XML file contains absolute paths, specify it to an empty string (`-b ""`).
- `-k`: specifies the installer kind, for instance most users will want **standard** here.
- `-o`: specifies the resulting installer Jar file name.

Figure 1.1: *The compiler architecture.*



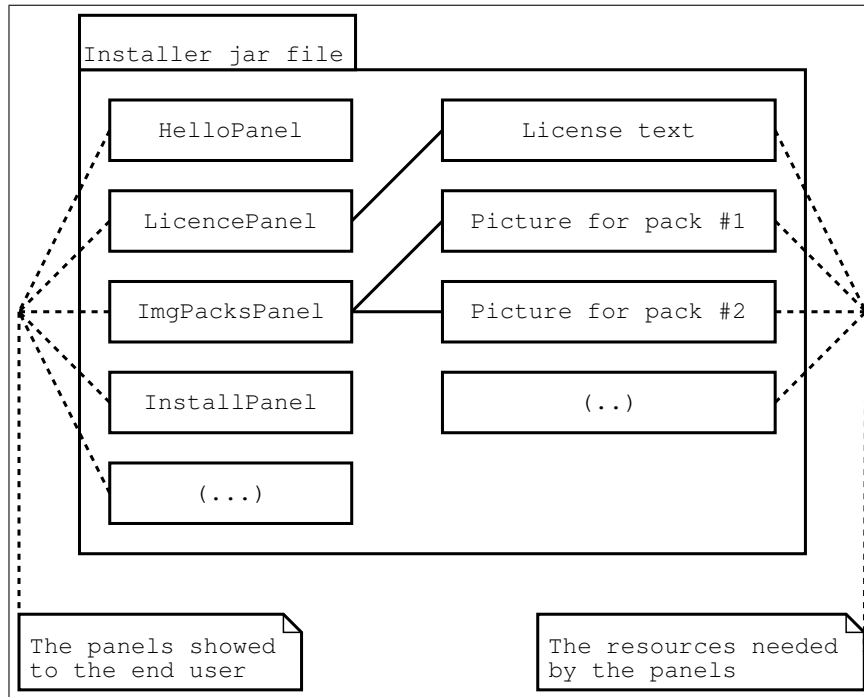
1.3.2 How an Installer Works

An installer presents its panels to the end-user. For instance, there is one to select the packages, one to prompt for the license agreement, one to select the installation path and so on. You have a choice from a variety of panels to place in the installer. For example, you can choose between a plain text and a HTML text panel for the license agreement. Also, if you don't want of the *HelloPanel*, you just don't include it.

It is very important to understand that some of the panels may need extra data. For instance, the license agreement panel needs the license text. A simple approach to specify such data would have been to add as many XML tags as needed for each panel. However, this makes the XML file too specific and not easy to maintain. The approach that has been chosen is to put the data in files and we call these files *resource files*. They are specified with a unique XML tag. This is a much cleaner approach.

You might wonder how your files are packaged. They can be grouped in *packs*. For instance, you can have one pack for the core files, one for the documentation, one for the source code and so on. In this way, your end-users

Figure 1.2: *The installer architecture.*



will have the choice to install a pack or not (provided that the pack they don't want to install is not mandatory). Inside the jar file (which is a zip file), a sub directory contains the pack files. Each pack file contains the files that are part of it. Could we do it simpler ? :-)

1.3.3 The Different Kinds of Installers

There are 2 kinds of installers available :

- **standard** : a single-file ready-to-run installer
- **web** : a web based installer (pack data is located on an HTTP server, and the installer retrieves it at install time (see section 3.6))

Chapter 2

Writing Installation XML Files

2.1 What You Need

2.1.1 Your editor

In order to write your XML installation files, you just need a plain text editor. Of course it's always easier to work with color coded text, so you might rather want to work with a text editor having such a feature. Here is a list of free editors that work well :

- Jext : <http://www.jext.org/>
- JEdit : <http://www.jedit.org/>
- classics like Vim and (X)Emacs.

2.1.2 Writing XML

Though you might not know much about XML, you have certainly heard about it. If you know XML you can skip this subsection as we will briefly present how to use XML.

XML is a markup language, really close to HTML. If you've ever worked with HTML the transition will be fast. However there are a few little things to know. The markups used in XML have the following form : `<markup>`. Each markup has to be closed somewhere with its ending tag : `</markup>`. Each tag can contain text and other markups. If a markup does not contain anything, it is just reported once : `<markup/>`. A markup can contain attributes like : `<markup attr1="123" attr2="hello !"/>`. Here is a sample of a valid XML structure :

```

<chapter title="Chapter 1">
  <section name="Introduction">
    <paragraph>
      This is the text of the paragraph number 1. It is available for the very low
      price of <price currency="dollar">1 000 000</price>.
    </paragraph>
  </section>
  <section name="xxx">
    xxx
  </section>
</chapter>

```

You should be aware of the following common mistakes :

- markups **are** case sensitive : `<markup>` is different from `<Markup>`.
- you **must** close the markups in the same order as you create them :
`<m1><m2>(...)</m2></m1>` is right but `<m1><m2>(...)</m1></m2>` is not.

Also, an XML file must start with the following header :

`<?xml version="1.0" encoding="iso-8859-1 standalone="yes" ?>`. The only thing you should modify is the encoding (put here the one your text editor saves your files to). The **standalone** attribute is not very important for us.

This (brief !) introduction to XML was just meant to enable you to write your installation specification. For a better introduction there are plenty of books and articles/tutorials dealing with XML on the Internet, in book stores, in magazines and so on.

2.2 Variable Substitution

During the installation process IzPack can substitute variables in various places with real values. Obvious targets for variable substitution are resource files and launch scripts, however you will notice many more places where it is more powerful to use variables rather than hard coded values. Wherever variables can be used it will be explained in the documentation.

There are three types of variables:

- Built-In variables. These are implemented in IzPack and are all dynamic in nature. This means that the value of each variable depends on local conditions on the target system.

- Environment variables. These are provided by the operating system the installer is run on.
- Variables that you can define. You also define the value, which is fixed for a given installation file.

You define your own variables in the installation XML file with the `<variable>` tag. How to do this is explained in detail later in this chapter.

Please note that when using variables they must always appear with a '\$' sign as the first character, even though they are not defined this way.

2.2.1 The Built-In Variables

The following variables are built-in :

- `$INSTALL_PATH` : the installation path on the target system, as chosen by the user
- `$JAVA_HOME` : the JavaTM virtual machine home path
- `$USER_HOME` : the user's home directory path
- `$USER_NAME` : the user name
- `$APP_NAME` : the application name
- `$APP_URL` : the application URL
- `$APP_VER` : the application version
- `$ISO3_LANG` : the ISO3 language code of the selected langpack.
- `$FILE_SEPARATOR` : the file separator on the installation system

2.2.2 Environment Variables

Environment variables can be accessed via the syntax `${ENV[variable]}`. The curly braces are mandatory. Note that variable names are case-sensitive and usually in UPPER CASE.

Example: To get the value of the OS environment variable "CATALINA_HOME", use `${ENV[CATALINA_HOME]}`.

2.2.3 Parse Types

Parse types apply only when replacing variables in text files. At places where it might be necessary to specify a parse type, the documentation will mention this. Depending on the parse type, IzPack will handle special cases -such as escaping control characters- correctly. The following parse types are available:

- **plain** - use this type for plain text files, where no special substitution rules apply. All variables will be replaced with their respective values as is.
- **javaprop** - use this type if the substitution happens in a Java properties file. Individual variables might be modified to function properly within the context of Java property files.
- **xml** - use this type if the substitution happens in a XML file. Individual variables might be modified to function properly within the context of XML files.
- **shell** - use this type if the substitution happens in a shell script. Because shell scripts use `$variable` themselves, an alternative variable marker is used: `%variable` or `%{variable}`.

2.3 The IzPack Elements

When writing your installer XML files, it's a good idea to have a look at the IZPACK installation DTD.

2.3.1 The Root Element <installation>

The root element of an installation is `<installation>`. It takes one required attribute : **version**. The attribute defines the version of the XML file layout and is used by the compiler to identify if it is compatible with the XML file. This should be set to 1.0 for the moment.

2.3.2 The Information Element <info>

This element is used to specify some general information for the installer. It contains the following elements :

- `<appname>` : the application name
- `<appversion>` : the application version
- `<appsubpath>` : the subpath for the default of the installation path. A variable substitution and a maskable slash-backslash conversion will be done. If this tag is not defined, the application name will be used instead.
- `<url>` : the application official website url
- `<authors>` : specifies the author(s) of the application. It must contain at least one `<author>` element whose attributes are :
 - `name` : the author's name
 - `email` : the author's email
- `<uninstaller>` : specifies whether to create an uninstaller after installation, it has only the `write` attribute, with default value `yes`. If this tag is not specified, the uninstaller will still be written.
- `<javaversion>` : specifies the minimum version of Java required to install your program. Values can be 1.2, 1.2.2, 1.4, etc. The test is a lexical comparison against the `java.version` System property on the install machine.
- `<webdir>` : Causes a “web installer” to be created, and specifies the URL packages are retrieved from at install time. The content of the tag must be a properly formed URL. See section 3.6 for more details.

Here is an example of a typical `<info>` section :

```
<info>
  <appname>Super extractor</appname>
  <appversion>2.1 beta 6</appversion>
  <appsubpath>myCompany/SExtractor</appsubpath>
  <url>http://www.superextractor.com/</url>
  <authors>
    <author name="John John Doo" email="jjd@jjd-mail.com"/>
    <author name="El Goyo" email="goyoman@mymail.org"/>
  </authors>
  <javaversion>1.2</javaversion>
</info>
```

2.3.3 The Variables Element <variables>

This element allows you to define variables for the variables substitution system. Some variables are built-in, such as `$INSTALL_PATH` (which is the installation path chosen by the user). When you define a set of variables, you just have to place as many `<variable>` tags in the file as needed. If you define a variable named `VERSION` you need to type `$VERSION` in the files to parse. The variable substitutor will then replace it with the correct value. One `<variable>` tag take the following attributes :

- `name` : the variable name
- `value` : the variable value

Here's a sample `<variables>` section :

```
<variables>
  <variable name="app-version" value="1.4"/>
  <variable name="released-on" value="08/03/2002"/>
</variables>
```

2.3.4 The GUI Preferences Element <guiprefs>

This element allows you to set the behavior of your installer GUI. This information will not have any effect on the command-line installers that will be available in future versions of IZPACK . The arguments to specify are :

- `resizable` : takes `yes` or `no` and indicates whether the window size can be changed or not.
- `width` : sets the initial window width
- `height` : sets the initial window height.

Here's a sample :

```
<guiprefs resizable="no" width="800" height="600"/>
```

Starting from IzPack 3.6, the look and feel can be specified in this section on a per-OS basis. For instance you can use the native look and feels on Win32 and OS X but use a third-party one on Unix-like platforms. To do that, you have to add some children to the `guiprefs` tag:

- **laf**: the tag that specifies a look and feel. It has a **name** parameter that defines the look and feel name.
- Each **laf** element needs at least one **os** tag, specified like in the other parts of the specification that support this tag.
- Like you can add **os** elements, you can add any number of **param** elements to customize a look and feel. A **param** elements has two attributes: **name** and **value**.

The available look and feels are:

- Kunststoff: **kunststoff**
- Liquid: **liquid**
- Metouia: **metouia**
- JGoodies Looks: **looks**

If you don't specify a look and feel for a particular operating system, then the default native one will be used: Windows on Windows, Aqua on Mac OS X and Metal on the Unix-like variants.

The *Liquid Look and Feel* supports the following parameters:

- **decorate.frames**: **yes** means that it will render the frames in Liquid style
- **decorate.dialogs**: **yes** means that it will render the dialogs in Liquid style

The *JGoodies Looks* look and feel can be specified by using the **variant** parameters. The values can be one of:

- **extwin**: use the Windows Extension look
- **plastic**: use the basic Plastic look
- **plastic3D**: use the Plastic 3D look
- **plasticXP**: use the Plastic XP look (default).

Here is a small sample:

```
<guiprefs height="600" resizable="yes" width="800">
  <laf name="metouia">
    <os family="unix" />
  </laf>
  <laf name="looks">
    <os family="windows" />
    <param name="variant" value="extwin" />
  </laf>
</guiprefs>
```

Starting from IzPack 3.7, some characteristics can be customized with the `<modifier>` tag which contains following attributes:

- **key**: a well defined key of the characteristic which should be changed.
- **value** the value for the key.

Following key value pairs are defined:

- **useButtonIcons**: possible are "yes" or "no". Default is "yes". If it is set to "no", all buttons which are created via the ButtonFactory contains no icon also a icon id was submitted. Directly created buttons are not affected.
- **useLabelIcons**: possible are "yes" or "no". Default is "yes". If it is set to "no", all labels which are created via the LabelFactory contains no icon also a icon id was submitted. Directly created labels are not affected.

2.3.5 The Localization Element `<locale>`

This element is used to specify the language packs (langpacks) that you want to use for your installer. You must set one `<langpack>` markup per language. This markup takes the `iso3` parameter which specifies the iso3 language code.

Here's a sample :

```

<locale>
  <langpack iso3="eng"/>
  <langpack iso3="fra"/>
  <langpack iso3="spa"/>
</locale>

```

The supported ISO3 codes are :

<i>ISO3 code</i>	<i>Language</i>
cat	Catalunyan
chn	Chinese
cze	Czech
dan	Danish
deu	German
eng	English
fin	Finnish
fra	French
hun	Hungarian
ita	Italian
jpn	Japanese
mys	Malaysian
ned	Nederlands
nor	Norwegian
pol	Polnish
por	Portuguese (Brazilian)
rom	Romanian
rus	Russian
scg	Serbian
spa	Spanish
svk	Slovakian
swe	Swedish
ukr	Ukrainian

2.3.6 The Resources Element <resources>

Several panels, such as the license panel and the shortcut panel, require additional data to perform their task. This data is supplied in the form of resources. This section describes how to specify them. Take a look at each panel description to see if it might need any resources. Currently, no checks are made to ensure resources needed by any panel have been included. The <resources> element is not required, and no <res> elements are required

within.

You have to set one `<res>` markup for each resource. Here are the attributes to specify :

- **src** : the path to the resource file which can be named freely of course (for instance `my-picture.jpg`).
- **id** : the resource id, depending on the needs of a particular panel
- **parse** : takes **yes** or **no** (default is **no**) - used to specify whether the resource must be parsed at the installer compilation time. For instance you could set the application version in a readme file used by `InfoPanel`.
- **type** : specifies the parse type. This makes sense only for a text resource - the default is **plain**, other values are **javaprop**, **xml** (Java properties file and XML files)
- **encoding** : specifies the resource encoding if the receiver needs to know. This makes sense only for a text resource.

Here's a sample :

```
<resources>
  <res id="InfoPanel.info" src="doc/readme.txt" parse="yes"/>
  <res id="LicencePanel.licence" src="legal/License.txt"/>
</resources>
```

2.3.7 The Panels Element `<panels>`

Here you tell the compiler which panels you want to use. They will appear in the installer in the order in which they are listed in your XML installation file. Take a look at the different panels in order to find the ones you need. The `<panel>` markup takes a single attribute **classname** which is the class-name of the panel.

Here's a sample :

```
<panels>
  <panel classname="HelloPanel"/>
  <panel classname="LicencePanel"/>
  <panel classname="TargetPanel"/>
  <panel classname="InstallPanel"/>
  <panel classname="FinishPanel"/>
</panels>
```


2.3.8 The Packs Element <packs>

This is a crucial section as it is used to specify the files that need to be installed. The <packs> section consists of several <pack> tags.

The <pack> takes the following attributes :

- **name**: the pack name
- **required**: takes **yes** or **no** and specifies whether the pack is optional or not.
- **os**: optional attribute that lets you make the pack targeted to a specific *operating system*, for instance **unix**, **mac** and so on.
- **preselected**: optional attribute that lets you choose whether the pack is by default selected for installation or not. Possible values are **yes** and **no**. A pack which is not preselected needs to be explicitly selected by the user during installation to get installed.
- **loose**: can be used so that the files are not located in the installer Jar. The possible values are **true** or **false**, the default being **false**. The author of this feature needed to put his application on a CD so that the users could run it directly from this media. However, he also wanted to offer them the possibility to install the software locally. Enabling this feature will make IzPack take the files on disk instead of from the installer. *Please make sure that your relative files paths are correct !*
- **id**: this attribute is used to give a unique id to the pack to be used for internationalization.

Internationalization of the PacksPanel

In order to provide internationalization for the PacksPanel, so that your users can be presented with a different name and description for each language you support, you have to create a file named **packsLang.xml.xyz** where **xyz** is the ISO3 code of the language in lowercase. Please be aware that case is significant. This file has to be inserted in the resources section of **install.xml** with the **id** and **src** attributes set at the name of the file. The format of these files is identical with the distribution langpack files located at **\$IZPACK_HOME/install/langpacks/installer**. For the name of the panel you just use the pack id as the txt id. For the description you use the pack id suffixed with **'.description'**.

The following sections describe the tags available for a <pack> section.

<description> - pack description

The contents of the **<description>** tag describe the pack contents. This description is displayed if the user highlights the pack during installation.

<depends> - pack dependencies

This can be used to make this pack selectable only to be installed only if some other is selected to be installed. The pack can depend on more than one by specifying more than one **<depends>** elements. Circular dependencies are not supported and the compiler reports an error if one occurs.

This tag takes the following attribute:

- **packname**: The name of the pack that it depends on

<os> - OS restrictions

It is possible to restrict a panel to a certain list of operating systems. This tag takes the following attributes:

- **family**: unix, windows or mac
- **name**: the exact OS name (ie Windows, Linux, ...)
- **version**: the exact OS version (see the JVM **os.version** property)
- **arch**: the machine architecture (see the JVM **os.arch** property).

<updatecheck>

This feature can update an already installed package, therefore removing superfluous files after installation. Here's how this feature author (Tino Schwarze) described it on the IzPack development mailing-list:

Each pack can now specify an **<updatecheck>** tag. It supports a subset of ant fileset syntax, e.g.:

```
<updatecheck>
  <include name="lib/**" />
  <exclude name="config/local/**" />
</updatecheck>
```

If the paths are relative, they will be matched relative to **\$INSTALL_PATH**. Update checks are only enabled if at least one **<include>** is specified. See **com.izforge.izpack.installer.Unpacker** for details.

<file> - add files or directories

The **<file>** tag specifies a file (a directory is a file too) to include into the pack. It takes the following attributes:

- **src**: the file location (relative path) - if this is a directory its content will be added recursively
- **targetdir**: the destination directory, could be something like `$INSTALL_PATH/subdirX`
- **os**: can optionally specify a target operating system (`unix`, `windows`, `mac`) - this means that the file will only be installed on its target operating system
- **override**: if `true` then if the file is already installed, it will be overwritten. Alternative values: `asktrue` and `askfalse` – ask the user what to do and supply default value for non-interactive use. Another possible values is `update`. It means that the new file is only installed if it's modification time is newer than the modification time of the already existing file (note that this is not a reliable mechanism for updates - you cannot detect whether a file was altered after installation this way.) By default it is set to `update`.

<additionaldata> This tag can also be specified in order to pass additional data related to a file tag for customizing.

- **<key>**: key to identify the data
- **<value>**: value which can be used by a custom action

<singlefile> - add a single file

Specifies a single file to include. The difference to **<file>** is that this tag allows the file to be renamed, therefore it has a **target** attribute instead of **targetdir**.

- **src**: the file location (relative path)
- **target**: the destination file name, could be something like `$INSTALL_PATH/subdirX/fileY`
- **os**: can optionally specify a target operating system (`unix`, `windows`, `mac`) - this means that the file will only be installed on its target operating system
- **override**: see **<file>** (2.3.8) for description

A **<additionaldata>** (2.3.8) tag can also be specified for customizing.

`<fileset>`: add a fileset

The `<fileset>` tag allows files to be specified using the powerful Jakarta Ant set syntax. It takes the following parameters:

- **dir**: the base directory for the fileset (relative path)
- **targetdir**: the destination path, works like for `<file>`
- **casesensitive**: optionally lets you specify if the names are case-sensitive or not - takes **yes** or **no**
- **defaultexcludes**: optionally lets you specify if the default excludes will be used - takes **yes** or **no**.
- **os**: specifies the operating system, works like for `<file>`
- **override**: see `<file>` for description
- **includes**: comma- or space-separated list of patterns of files that must be included; all files are included when omitted. This is an alternative for multiple include tags.
- **excludes**: comma- or space-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted. This is an alternative for multiple exclude tags.

You specify the files with `<include>` and `<exclude>` tags that take the **name** parameter to specify the Ant-like pattern :

- ****** : means any subdirectory
- ***** : used as a wildcard.

Here are some examples of Ant patterns :

- `<include name="lib"/>` : will include **lib** and the subdirectories of **lib**
- `<exclude name="**/*.java"/>` : will exclude any file in any directory starting from the base path ending by **.java**
- `<include name="lib/*.jar"/>` : will include all the files ending by **.jar** in **lib**
- `<exclude name="lib/**/*F00*"/>` : will exclude any file in any subdirectory starting from **lib** whose name contains **F00**.

There are a set of definitions that are excluded by default file-sets, just as in Ant. IzPack defaults to the Ant list of default excludes. There is currently no equivalent to the `defaultexcludes` task. Default excludes are:

```
**/*\~{}  
**/*\#*\#  
**/*.\#*  
**/*%*%  
**/*.\_*  
**/*CVS  
**/*CVS/**  
**/*cvsignore  
**/*SCCS  
**/*SCCS/**  
**/*vssver.scc  
**/*svn  
**/*svn/**  
**/*DS_Store
```

A `<additionaldata>` ([2.3.8](#)) tag can also be specified for customizing.

`<parsable>` - parse a file after installation

Files specified by `<parsable>` are parsed after installation and may have variables substituted.

- **targetfile** : the file to parse, could be something like `$INSTALL_PATH/bin/launch-script.sh`
A slash will be changed to the system dependant path separator (e.g. to a backslash on Windows) only if no backslash masks the slash.
- **type** : specifies the type (same as for the resources) - the default is `plain`
- **encoding** : specifies the file encoding
- **os**: specifies the operating system, works like for `<file>`

`<executable>` - mark file executable or execute it

The `<executable>` tag is a very useful thing if you need to execute something during the installation process. It can also be used to set the executable flag on Unix-like systems. Here are the attributes :

- **targetfile** : the file to run, could be something like
`$INSTALL_PATH/bin/launch-script.sh`
 Slashes are handled special (see attribute **targetfile** of tag `<parsable>`[2.3.8](#)).
- **class** : If the executable is a jar file, this is the class to run for a Java™ program
- **type** : `bin` or `jar` (the default is `bin`)
- **stage** : specifies when to launch : `postinstall` is just after the installation is done and the default value, `never` will never launch it (useful to set the `+x` flag on Unix). `uninstall` will launch the executable when the application is uninstalled. The executable is executed before any files are deleted.
- **failure** : specifies what to do when an error occurs : `abort` will abort the installation process, `ask` (default) will ask the user what to do and `warn` will just tell the user that something is wrong
- **os**: specifies the operating system, works like for `<file>`
- **keep** : specifies whether the file will be kept after execution. The default is to delete the file after it has been executed. This can be changed by specifying `keep="true"`.

A `<args>` tag can also be specified in order to pass arguments to the executable:

- `<arg>`: passes the argument specified in the `value` attribute. Slashes are handled special (see attribute **targetfile** of tag `<parsable>`[2.3.8](#)).

`<os>` - make a file OS-dependent

The `<os>` tag can be used inside the `<file>`, `<fileset>`, `<singlefile>`, `<parsable>`, `<executable>` tags to restrict its effect to a specific operating system family, architecture or version:

- **family**: `unix`, `windows`, `mac` to specify the operating system family
- **name**: the operating system name
- **version**: the operating system version
- **arch**: the operating system architecture (for instance the Linux kernel can run on `i386`, `sparc`, and so on)

Here's an example installation file :

```
<packs>
  <!-- The core files -->
  <pack name="Core" required="yes">
    <description>The IzPack core files.</description>
    <file targetdir="$INSTALL_PATH" src="bin"/>
    <file targetdir="$INSTALL_PATH" src="lib"/>
    <file targetdir="$INSTALL_PATH" src="legal"/>
    <file targetdir="$INSTALL_PATH" src="Readme.txt"/>
    <file targetdir="$INSTALL_PATH" src="Versions.txt"/>
    <file targetdir="$INSTALL_PATH" src="Thanks.txt"/>
    <parsable targetfile="$INSTALL_PATH/bin/izpack-fe"/>
    <parsable targetfile="$INSTALL_PATH/bin/izpack-fe.bat"/>
    <parsable targetfile="$INSTALL_PATH/bin/compile"/>
    <parsable targetfile="$INSTALL_PATH/bin/compile.bat"/>
    <executable targetfile="$INSTALL_PATH/bin/compile" stage="never"/>
    <executable targetfile="$INSTALL_PATH/bin/izpack-fe" stage="never"/>
  </pack>

  <!-- The documentation (1 directory) -->
  <pack name="Documentation" required="no">
    <description>The IzPack documentation (HTML and PDF).</description>
    <file targetdir="$INSTALL_PATH" src="doc"/>
  </pack>
</packs>
```

2.3.9 The Native Element <native>

Use this if you want to use a feature that requires a native library. The native libraries are placed under `bin/native/...`. There are 2 kinds of native libraries : the IzPACK libraries and the third-party ones. The IzPack libraries are located at `bin/native/izpack`, you can place your own libraries at `bin/native/3rdparty`. It is possible to place a native library also into the uninstaller. It is useable from CustomActions (7). If one or more are referenced for it, the needed support classes are automatically placed into the uninstaller. To place it only on operating systems for which they are build, it is possible to define an OS restriction. This restriction will only be performed for the uninstaller. The markup takes the following attributes :

- **type** : `izpack` or `3rdparty`
- **name** : the library filename
- **stage**: stage where to use the library (install—uninstall—both)

<os> - make a library OS-dependent

The `<os>` tag can be used to restrict the inclusion into the uninstaller to a specific operating system family, architecture or version. The inclusion into the installer will be always done. For more information see [2.3.8](#).

Here's a sample :

```
<native type="izpack" name="ShellLink.dll"/>
```

2.3.10 The Jar Merging Element <jar>

If you adapt IZPACK for your own needs, you might need to merge the content of another jar file into the jar installer. For instance, this could be a library that you need to merge. The `<jar>` markup allows you to merge the raw content of another jar file into the installer and the uninstaller. It is necessary that the paths in the jars are unique because only the contained files of the jar are added to the installer jar, not the jar file self. The attributes are:

- **src** : the path at compile time
- **stage**: stage where to use the contents of the additional jar file (install—uninstall—both)

A sample :

```
<jar src="../../nicelibrary.jar"/>
```

2.4 The Available Panels

In this section I will introduce the various panels available in IzPack. The usage for most is pretty simple and described right here. The more elaborate ones are explained in more detail in the *Advanced Features* chapter or in their own chapter. The panels are listed by their class name. This is the name that must be used with the `classname` attribute (case-sensitive).

2.4.1 HelloPanel

This panel welcomes the user by displaying the project name, the version, the URL as well as the authors.

2.4.2 InfoPanel and HTMLInfoPanel

This is a kind of 'README' panel. It presents text of any length. The text is specified by the `(HTML)InfoPanel.info` resource. Starting from IzPack 3.7.0, variables substitution is allowed.

2.4.3 LicencePanel and HTMLLicencePanel

*Note : there is a mistake in the name - it should be `LicensePanel`. In France the word is *Licence* ... and one of my diploma is a '*Licence*' so ... :-)*

These panels can prompt the user to acknowledge a license agreement. They block unless the user selects the 'agree' option. To specify the license agreement text you have to use the `(HTML)LicencePanel.licence` resource.

2.4.4 PacksPanel

Allows the user to select the packs he wants to install.

2.4.5 ImgPacksPanel

This is the same as above, but for each panel a different picture is shown to the user. The pictures are specified with the resources `ImgPacksPanel.img.x` where x stands for the pack number, the numbers start from 0. Of course it's up to you to specify as many images as needed and with correct numbers. For instance if you have 2 packs `core` and `documentation` (in this order), then the resource for `core` will be `ImgPacksPanel.img.0` and the resource for `doc` will be `ImgPacksPanel.img.1`. The supported image formats depend on what your JVM supports, but starting from J2SE 1.3, *GIF*, *JPEG* and *PNG* are supported.

2.4.6 TargetPanel

This panel allows the user to select the installation path. It can be customized with the following resources (they are text files containing the path) :

- `TargetPanel.dir.f` where f stands for the family (`mac`, `macosx`, `windows`, `unix`)

- `TargetPanel.dir` : the directory name, instead of the software to install name
- `TargetPanel.dir.d` where `d` is a "dynamic" name, as returned by the JavaTM virtual machine. You should write the name in lower-case and replace the spaces with underscores. For instance, you might want a different setting for Solaris and GNU/Linux which are both Unix-like systems. The resources would be `TargetPanel.dir.sunos`, `TargetPanel.dir.linux`. You should have a Unix-resource in case it wouldn't work though.

2.4.7 InstallPanel

You should always have this one as it launches the installation process !

2.4.8 XInfoPanel

A panel showing text parsed by the variable substitutor. The text can be specified through the `XInfoPanel.info` resource. This panel can be useful when you have to show information after the installation process is completed (for instance if the text contains the target path).

2.4.9 FinishPanel

A ending panel, able to write automated installer information. For details see the chapter on 'Advanced Features'.

2.4.10 SimpleFinishPanel

Same as `FinishPanel`, but without the automated installer features. It is aimed at making the life easier for end-users who will never encounter the automated installer extra feature.

2.4.11 ShortcutPanel

This panel is used to create desktop shortcuts. For details on using the ShortcutPanel see the chapter 'Desktop Shortcuts'.

2.4.12 UserInputPanel

This panel allows you to prompt the user for data. What the user is prompted for is specified using an XML file which is included as a resource to the installer. See chapter 6 on page 69 for a detailed explanation.

2.4.13 CompilePanel

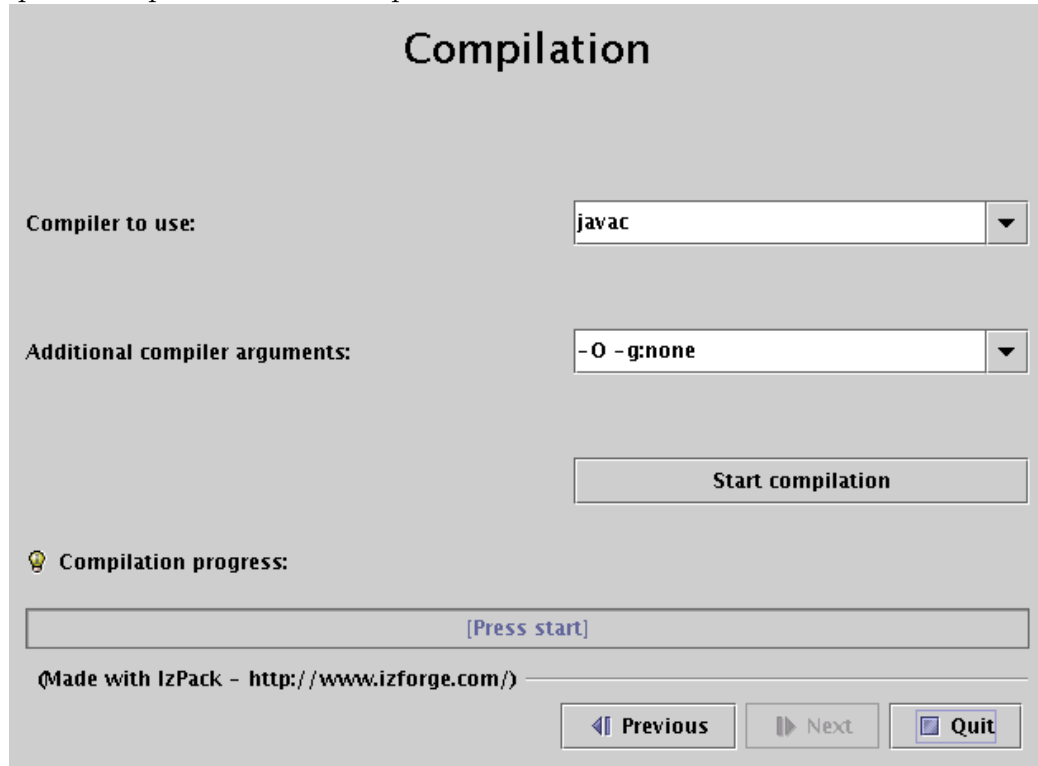
This panel allows you to compile just installed Java sourcecode. The details for the compilation are specified using the resource `CompilePanel.Spec.xml`. The XML file has the following format:

```
<compilation>
  <global>
    <compiler>
      <choice value="$JAVA_HOME/bin/javac" />
      <choice value="jikes" />
    </compiler>
    <arguments>
      <choice value="-O -g:none" />
      <choice value="-O" />
      <choice value="-g" />
      <choice value="" />
    </arguments>
  </global>
  <jobs>
    <classpath add="$INSTALL_PATH/src/classes/" />
    <job name="optional name">
      <directory name="$INSTALL_PATH/src/classes/xyz" />
    </job>
    <job name="another job">
      <packdependency name="some package name" />
      <classpath sub="$INSTALL_PATH/" />
      <directory name="$INSTALL_PATH/src/classes/abc" />
      <file name="$INSTALL_PATH/some/file.java" />
    </job>
  </jobs>
</compilation>
```

</compilation>

In theory, jobs can be nested but this has not been tested at all. A change to the classpath within a job only affects this job and nested jobs. The classpath should be specified before any files or directories.

The user can change the compiler to use and choose from some default compilation options before compilation is started.



2.4.14 ProcessPanel

This panel allows you to execute arbitrary files after installation. The details for the compilation are specified using the resource `ProcessPanel.Spec.xml`.

The XML file has the following format:

```
<processing>
  <job name="do xyz">
    <os family="windows" />
    <executefile name="$INSTALL_PATH/scripts/xyz.bat">
      <arg>doit</arg><arg>$variable</arg>
    </executefile>
  </job>
```

```

<job name="do xyz">
  <os family="unix" />
  <executefile name="$INSTALL_PATH/scripts/xyz.sh">
    <arg>doit</arg><arg>$variable</arg>
  </executefile>
</job>
</processing>

```

Each job may have an `<os>` attribute – see [2.3.8](#) for details.

It is also possible to execute Java classes from this panel. Here's what this feature author (Alex Bradley) says:

I've been able to work around my requirements by extending the `ProcessPanelWorker` functionality to run user-specified classes. I've extended the DTD of the `ProcessPanel.Spec.xml` to include a new element:

```

<executableclass name="classname">
  <args..../>
</executableclass>

```

I've also added a new sub-class of `Processable` called `ExecutableClass`. This will run a user-specified class in the context of the installer JVM with a single method :

```
run( AbstractUIHandler handler, String[] args);
```

It can do everything I need and more. In particular, it allows me to write a process extension and still be able to provide feedback to the user through the feedback panel, and to add new functionality to the installer, after its been built.

New with version 3.7 is the possibility to tee output that is written to the `ProcessPanel`'s textarea into an optional logfile. Using this feature is pretty much straightforward, you only have to add a line in `ProcessPanel.Spec.xml` that will tell `IzPack` the location, where the logfile should be stored.

Variable substitution is performed, so you can use `$INSTALL_PATH` as example.

The name of the logfile is not (yet) configurable but should fit in most cases. It will be named

```
Install_V<$APP_VER>_<YYYY>-<MM>-<DD>_<hh>-<mm>-<ss>_<RandomId>.log
```

Here's an example:

```
<processing>
  <logfiledir>$INSTALL_PATH</logfiledir>
  <job name="do xyz">
    <os family="windows" />
    <executefile name="$INSTALL_PATH/scripts/xyz.bat">
      <arg>doit</arg><arg>$variable</arg>
    </executefile>
  </job>
</processing>
```

This will generate a logfile named e.g. `Install_V1.3_2004-11-08_19-22-20_43423.log` located in `$INSTALL_PATH`.

`ProcessPanelWorker` will write all output that is directed to `stdout` and `stderr` to this file if `ProcessPanel.Spec.xml` contains the `logfiledir` entry.

Please note that this one file is used for storing the complete output of all jobs and not a file for each job that is run.

2.4.15 JDKPathPanel

This panel allows the user to select a JDK path. The variable `JAVA_HOME` does not point to a JDK, else to a JRE also the environment variable points to a JDK. This is not a bug, this is the behavior of the VM. But some products needs a JDK, for that this panel can be used. There is not only a selection of the path else a validation. The validation will be done with the file `JDKPath/lib/tools.jar`. If `JAVA_HOME` points to the VM which is placed in the JDK, the directory will be used as default (`JAVA_HOME/..`). If there is the variable

`JDKPathPanel.skipIfValid`

defined with the value "yes", the panel will be skipped if the path is valid. Additional it is possible to make a version control. If one or both variables

`JDKPathPanel.minVersion`

`JDKPathPanel.maxVersion`

are defined, only a JDK will be accepted which has a version in the range of it. The detection is a little bit pragmatically, therefor it is possible, that the detection can fail at some VMs. The values in the `install.xml` should be like

```
<variables>
  <variable name="JDKPathPanel.minVersion" value="1.4.1" />
  <variable name="JDKPathPanel.maxVersion" value="1.4.99" />
  <variable name="JDKPathPanel.skipIfValid" value="yes" />
</variables>
```

If all is valid, the panels `isValidated` method sets the variable

`JDKPath`

to the chosen path. Be aware, this variable exist not until the `JDKPanel` was quitted once. At a secound activation, the default will be the last selection.

Chapter 3

Advanced Features

3.1 Ant Integration

IzPACK can be easily integrated inside an Ant build process. To do so you first need to tell Ant that you would like to use IzPACK :

```
<!-- Allows us to use the IzPack Ant task -->
<taskdef name="izpack" classpath="${basedir}/lib/compiler.jar"
         classname="com.izforge.izpack.ant.IzPackTask"/>
```

If you want to use the standalone compiler (and therefore don't need an IzPack installation for building), the task needs to be defined as follows:

```
<!-- Allows us to use the IzPack Ant task -->
<taskdef name="izpack" classpath="${basedir}/lib/standalone-compiler.jar"
         classname="com.izforge.izpack.ant.IzPackTask"/>
```

Don't forget to add `compiler.jar` or `standalone-compiler.jar` to the classpath of the Ant process.

Then you can invoke IzPACK with the `izpack` task which takes the following parameters:

- `input` : the XML installation file
- `output` : the output jar installer file
- `installerType` : optional. `standard` or `web`. If `web`, the `<webdir>` attribute must be specified in the input file (see section 3.6). Used to force creation of a standard installer when the `<webdir>` attribute has been used.
- `baseDir` : the base directory to resolve the relative paths

- `izPackDir`: the IZPACK home directory. Only necessary if you do not use the standalone compiler.

Here is a sample of the task invocation:

```
<!-- We call IzPack -->
<echo message="Makes the installer using IzPack"/>
<izpack input="${dist.dir}/IzPack-install.xml"
        output="${dist.dir}/IzPack-install.jar"
        installerType="standard"
        basedir="${dist.dir}"
        izPackDir="${dist.dir}"/>
```

3.2 System properties as variable

All system properties are available as `$_SYSTEM_<variable>` where `<variable>` is the actual name `_BUT_` with all separators replaced by `'_'`. Properties with null values are never stored.

Examples:

`$_SYSTEM_java_version` or `$_SYSTEM_os_name`

3.3 Automated Installers

When you conclude your installation with a `FinishPanel`, the user can save the data for an automatic installation. With this data, he will be able to run the same installation on another similar machine. In an environment where many computers need to be supported this can save *a lot* of time.

So run once the installation on a machine and save your automatic installation data in `auto-install.xml` (that's just a sample). Then put this file in the same directory as the installer on another machine. Run it with:

```
java -jar installer.jar auto-install.xml
```

It has reproduced the same installation :-)

3.4 Picture on the Language Selection Dialog

You can add a picture on the language selection dialog by adding the following resource : `installer.langsel.img`. *GIF*, *JPEG* and *PNG* pictures are supported starting from J2SE 1.3.

3.5 Picture in the installer

It is possible to specify an optional picture to display on the left side of the installer. To do this, you just have to define a resource whose id is `Installer.image`. For instance,

```
<res id="Installer.image" src="nice-image.png" />
```

will do that. If the resource isn't specified, no picture will be displayed at all. *GIF*, *JPEG* and *PNG* pictures are supported starting from J2SE 1.3.

You can also give a specific picture for a specific panel by using the `Installer.image.n` resource names where *n* is the panel index. For instance if you want a specific picture for the third panel, use `Installer.image.2` since the indexes start from 0.

3.6 Web Installers

The web installers allow your users to download a small installer that does not contain the files to install. These files will be downloaded from an HTTP server such as *Apache HTTPD*. If you have many optional packs, this can save people's resources. It's very easy: people download a small Jar file containing the installer, they launch it and choose their packages. Then the installer will get the required packages from individual Jar files located on a server, only downloading those required. It's that simple.

To create a web installer, add the `<webdir>` element to the `<info>` element (see section [2.3.2](#)). The text must be a valid, fully qualified URL for a directory on the web server.

```
<info>  
  <appname>Super extractor</appname>
```

```
<appversion>2.1 beta 6</appversion>
<url>http://www.superextractor.com/</url>
<webdir>http://www.superextractor.com/download</url>
</info>
```

You can force creation of a standard installer even if `webdir` is specified, by specifically creating a `standard` installer from the command line invocation or ant task (see).

When installing, if the user is behind a firewall, attempting download the jar files may fail. If this happens, the user will be prompted to enter the name hostname and port of their firewall.

You may password protect the files using mechanisms provided by your web server, IzPack will prompt for a password at install time, when required.

3.7 More Internationalization

3.7.1 Special resources

IzPack is available in several languages. However you might want to internationalize some additional parts of your installer. In particular you might want this for the `*InfoPanel` and `*LicencePanel`. This is actually pretty easy to do. You just have to add one resource per localization, suffixed with the ISO3 language code. At runtime these panels will try to load a localized version.

For instance let's suppose that we use a `HTMLInfoPanel`. Suppose that we have it in English, French and German. We want to have a French text for french users. Here we add a resource pointing to the French text whose name is `HTMLInfoPanel.info_fra`. And that's it ! English and German users (or anywhere other than in France) will get the default text (denoted by `HTMLInfoPanel.info`) and the French users will get the French version. Same thing for the other Licence and Info panels.

To sum up : add `<iso3 code>` to the resource name for `InfoPanel`, `HTMLInfoPanel`, `LicencePanel` and `HTMLLicencePanel`.

3.7.2 Packs

Thanks to Thorsten Kamann, it is possible to translate the packs names and descriptions. To do that, you have to define a special identifier in the ele-

ments of the XML installation file and add the related entries in the suitable langpacks. For instance if you have the following XML snippet:

```
<pack name="core" id="core.package" ...>  
  <description/>  
</pack>
```

then the related entries of the langpacks will look like this:

```
<str id="core.package" txt="Core Package"/>  
<str id="core.package.description" txt="The core package provides
```

Chapter 4

Desktop Shortcuts

(by Elmar GROM and Marc EPPELMANN)

4.1 Defining Shortcuts

4.1.1 Introduction

On today's GUI oriented operating systems, users are used to launching applications, view web sites, look at documentation and perform a variety of other tasks, by simply clicking on an icon on the desktop or in a menu system located on the desktop. Depending on the operating system these icons have different names. In this context we will refer to them collectively as shortcuts.

Apart from actually placing an application on the target system, users routinely expect an installer to create the necessary shortcuts for the application as well. For you as application developer, this means that for a professional appearance of your product you should also consider creating shortcuts.

In contrast to the general specification of an IzPack installer, the specification of shortcuts in IzPack requires a little more effort. In addition, some of the concepts are a bit more complex and there are some operating system specific issues to observe. Fortunately, you only need to worry about operating system specifics if you want to deploy your application to multiple different operating systems. In any case, it will pay off to spend some time to study this documentation and the example spec files before you start to implement your own shortcuts.

At the time of writing this Chapter the current IzPack Version 3.7.0-M3 is only capable to creating shortcuts on

1. Microsoft Windows
and
2. Unix and Unix-based operating systems (like Linux), which use the [X11](#) GUI-System and [FreeDesktop.org](#) based shortcut handling (such as [KDE](#) and [Gnome](#)).

Other operating or GUI systems, such as MacOS < MacOS-X are not supported. However, there is a special UI-variant that automatically pops up on unsupported systems. It informs the user about the intended targets of your shortcuts and allows the user to save this information to a text file. While this is not an elegant solution, at least it aids the user in the manual creation of the shortcuts.

If you would like to review what an end user would see if the target operating system is not supported, you can do the following. Simply place the tag `<notSupported/>` in the spec file. This tag requires no attributes or other data. It must be placed under `<shortcuts>`, just like the individual shortcut specifications. Be sure to remove this tag before getting your application ready for shipment.

We expect other operating systems to be supported in the near future and as always, contributions are very welcome. At present someone is actively working on Mac support.

4.1.2 What to Add to the Installer

There are some things that you have to add to your installer to enable shortcut creation. Obviously you need to add the panel responsible for creating shortcuts. This panel is aptly enough called `ShortcutPanel`. However, in order for the `ShortcutPanel` to work properly a number of additional items are required. These must be added manually to the installer, as all other resources, since the front-end will be rewritten. In this chapter we will explain which of these items are required and for what reason.

First, we would like to discuss items that are supplied with IzPack and only need to be added to the installer. After that, we move on to the things you have to prepare yourself before you can add them. The way in which

shortcuts are created varies widely among operating systems. In some cases it is possible to do this with pure Java code, while other systems -such as MS-Windows- require native code to accomplish this task. On the other side, the current implementation, which creates shortcuts on Unix based systems needs no native library at all, since it works with 'these' pure Java code. The native library required for the Windows operating systems are supplied with IzPack is called **ShellLink.dll**. Note: They will not be automatically added to your installer file. You need to list them yourself in the XML file for the installer. A description how to do this follows in the next section.

Native libraries can be added to the installer by using the `<native>` tag. To add the **ShellLink.dll**, you just have to add the following line to the installer XML file:

```
<native type="izpack" name="ShellLink.dll"/>
```

For more details about the use of the `<native>` tag see the chapter about the format of the XML file.

You have also to add an extra specification file for each platform family to enable shortcut creation on these platforms. At least one (the default file) is required by the shortcut panel. The format of all spec files is XML and they must be added to the installer as a resource. The source name of this specification does not matter, however its resource name (also called id or alias) when added to the installer must be **(prefix)+shortcutSpec.xml**. At this release, there are only two prefixes supported: "Win_" for the Windows family and "Unix_" for all Unixes. If the prefix is omitted the shortcut panel searches for a named resource: **shortcutSpec.xml**. This is the default resource name. As the default resource name will be used on Windows platforms, the "Win_shortcutSpec.xml" can be omitted.

Hint: If the shortcut panel does not find one of these named resources, it will never appears. So, do not use different resource names and do not add a path to these.

Example

```
<res src="C:\MyDocuments\Installer\default_shortcut_specification.xml"
    id="shortcutSpec.xml"/>
<res src="C:\MyDocuments\Installer\unix_shortcut_specification.xml"
    id="Unix_shortcutSpec.xml"/>
```

Why use different shortcut spec files?

1. The Target filenames are most different.(batch files on Windows vs. shell scripts on Unix.)
2. The Icon file formats are different. ICOs on Windows-, PNGs on Unix-platforms.
3. The Target locations can be different.

This is the simple reason.

4.1.3 Why Native Code to do the Job on Windows?

by Elmar

This little chapter is not strictly part of the documentation but I have been asked this question sufficiently often that I think it's worth explaining right here. It is certainly a natural question to ask. After all IzPack is an application completely written in Java and primarily targeted for the installation of Java based programs. So why wouldn't we try to keep everything pure Java and avoid the use of native code altogether? There must be some personal preference of the developer hidden behind this approach you might think. Well, not really, but I admit at first it seems quite feasible to write it all in Java. On virtually any operating system or GUI surface around, Shortcuts are simply files on the local file system. Files can be created and accessed directly from within Java, so why should there be a need for using native code?

Well, it turns out that just creating a file is not good enough, it also needs to have the right content. Shell Links as they are called in Windows land are binary files. I actually managed to find documentation on the format. Naturally this was hacker data, you won't get this sort of thing from Microsoft (by the way: thanks a lot to Jesse Hager for a smash job!). Armed with this information I tried to create these files myself in Java. The problem was that the documentation was not entirely accurate and had some gaps as well. I tried for over a month to get this to work but finally I had to give up. Even if I would have succeeded, it would have been a hack, since a shell link requires some information that is impossible to obtain from within Java. Usually you can successfully create a shell link by only filling in the bare minimum information and then ask Windows to resolve the link. Windows then repairs the shell link. Unfortunately this was only the beginning, soon I encountered a host of other problems. For one thing, the installer needs to know the correct

directories for placing the links and it turns out they are named differently in different countries. In addition, there are ways of manually modifying them, which some people might actually have done. The only way to place the shortcut files reliably is through accessing the Windows Registry. Naturally, this operation also required native code. Same thing with asking Windows to resolve the link... On the bottom line, at every step and turn you run into an issue where you just need to use native code to do the trick. So I decided that I would do it the proper way all the way through. That is in a nutshell the reason why I used native code to create shortcuts on MS-Windows.

As I am writing this I am at work with a friend to replicate this work for the Mac and it looks very much like we need to take the same approach there as well. On the various Unix GUIs on the other hand, we are lucky that we can do the job without native libraries.

4.1.4 The Shortcut Specification

As we say above, the specification for shortcuts is provided to the Shortcut-Panel in the XML fileformat. At the time of this writing (for IzPack version 3.7.0-M3) the front-end will be rewritten. Until these work is in progress you have to write the specification files manually. For your convenience, there are two annotated sample specification files in the sample subdirectory of your IzPack installation. At the beginning you might want to experiment with these files.

Both specification files have one root element called `<shortcuts>`. This root elements recognizes 3 different child elements: `<programGroup>`, `<skipIfNotSupported/>` and `<shortcut>`.

`<skipIfNotSupported/>` can be used to avoid the panel to show the alternative UI which shows the shortcut information that would have been created on a system that supports it. In other words, using this tag will make the panel be silent on non-supported systems. The default is to show the alternative UI.

The `<programGroup>` tag allows you to specify the name of the menu, or more precise, the folder in which the shortcuts will be grouped. The exact location and appearance of the program group depends on the specific target system on which the application will be installed, however you can partially control it. Please note that `<programGroup>` may only appear once

in the specification. If more than one instance occurs, only the first one will be used. This tag requires two attributes: **defaultName** and **location**. **defaultName** specifies the name that the group menu should have on the target system. You should be aware that the ShortcutPanel will present this name to the user as a choice. The user can then edit this name or select a group that already exists. As a result, there is no guarantee that the actual name of the program group on the target system is identical with your specification. **location** specifies where the group menu should show up. There are two choices: **applications** and **startMenu**. If you use **applications**, then the menu will be placed in the menu that is ordinarily used for application shortcuts. **applications** is recommended for Unix shortcuts. If you use **startMenu**, the group menu will be placed at the top most menu level available on the target system. Depending on the target system, it might not be possible to honor this specification exactly. In such cases, the ShortcutPanel will map the choice to the location that most closely resembles your choice. Unix shortcuts do not need to support the **startMenu**, because the **applications** menu is already on the highest level. This means this has no affect on the platform.

For each shortcut you want to create, you have to add one `<shortcut>` tag. Most details about the shortcut are listed as attributes with this tag. The following sections describe what each attribute does, which attributes are optional and which ones are required and what the values are that are accepted for each of the attributes. Note that all attributes that have a yes/no choice can also be omitted. Doing so has the same effect as using a value of no. The shortcut attributes can be divided into two groups

- attributes that describe properties of the shortcut
- attributes that define the location(s) at which a copy of the shortcut should be placed.

The following attributes are used to define location:

- **programGroup**
- **desktop**
- **applications**
- **startMenu**

- startup

4.1.5 Shortcut Attributes

There are three classes of attributes. Some are required, most are completely optional and some are semi-optional. The set of semi-optional attributes are all the attributes used to define the location of a shortcut. These are semi-optional because for any individual one it is your choice if you want to include it or not. However they are not completely optional. You must specify at least one location. If all were omitted, the instruction would essentially tell the panel that a copy of this shortcut is to be placed at no location. In other words no copy is to be placed anywhere.

name - required

The value of this attribute defines the name that the shortcut will have. This is the text that makes up the menu name if the shortcut is placed in a menu or the caption that is displayed with the shortcut if it is placed on the desktop.

target - required

The value of this attribute points to the application that should be launched when the shortcut is clicked. The value is translated through the variable substitutor. Therefore variables such as `$INSTALL_PATH` can be used to describe the location. **You should be aware that the use of this tag is likely to change once other operating systems are supported.**

commandLine - optional

The value of this attribute will be passed to the application as command line. I recommend to work without command line arguments, since these are not supported by all operating systems. As a result, your applications will not be portable if they depend on command line arguments. Instead, consider using system properties or configuration files.

workingDirectory - optional

This attribute defines the working directory for the application at the time it is launched. I would recommend some caution in relying on this

too heavily if your application should be portable, since this might not be supported by all operating systems. At this time I don't have enough information to make a definite statement one way or the other. The value is translated through the variable substitutor. Therefore variables such as `$INSTALL_PATH` can be used to describe the directory.

description - optional

The value of this attribute will be visible to the user when a brief description about associated application is requested. The form of the request and the way in which this description is displayed varies between operating systems. On MS-Windows the description is shown as a tool tip when the mouse cursor hovers over the icon for a few seconds. On some operating systems this feature might not be supported but I think it is always a good idea to include a brief description.

iconFile - optional

The value of this attribute points to the file that holds the icon that should be displayed as a symbol for this shortcut. This value is also translated through the variable substitutor and consequently can contain variables such as `$INSTALL_PATH`. If this attribute is omitted, no icon will be specified for the shortcut. Usually this causes the OS to display an OS supplied default icon. **The use of this attribute is also likely to change once other operating systems are supported. Read the Section about Icons below, for more information.**

iconIndex - optional

If the file type for the icon supports multiple icons in one file, then this attribute may be used to specify the correct index for the icon. I would also advise against using this feature, because of operating system incompatibilities in this area. In file formats that do not support multiple icons, this values is ignored.

initialState - optional

There are four values accepted for this attribute: `noShow`, `normal`, `maximized` and `minimized`. If the target operating system supports this feature, then this value will have the appropriate influence on the initial window state of the application. `noShow` is particularly useful when launch scripts are used

that cause a command window to open, because the command window will not be visible with this option. For instance on MS-Windows starting a batch file that launches a Java application has the less than pretty side effect that two windows show: the DOS command prompt and the Java application window. Even if the shortcut is configured to show minimized, there are buttons for both windows in the task bar. Using **noShow** will completely eliminate this effect, only the Java application window will be visible. *On Unix use **normal** , because this is not supported.*

programGroup - semi-optional

The value for this attribute can be either yes or no. Any other value will be interpreted as no. If the value is yes, then a copy of this shortcut will be placed in the group menu. *On Unix (KDE) this will always be placed on the top level.*

desktop - semi-optional

For this attribute the value should also be yes or no. If the value is yes, then a copy of the shortcut is placed on the desktop. *On Unix the shortcuts will only be placed on the (KDE-) desktop of the user, who currently runs the installer. For Gnome the user can simply copy the *.desktop files from `~Desktop` to `~gnome-desktop`. (This is already a TODO for the Unix-shortcut implementation.)*

applications - semi-optional

This is also a yes/no attribute. If the value is yes, then a copy of the shortcut is placed in the applications menu (if the target operating system supports this). This is the same location as the applications choice for the program group. *This makes no sense on Unix.*

startMenu - semi-optional

This is a yes/no attribute as well. If the value is yes, then a copy of the shortcut is placed directly in the top most menu that is available for placing application shortcuts. *This is not supported on Unix. see above.*

startup - semi-optional

This is also a yes/no attribute. If the value is yes, then a copy of the short-

cut is placed in a location where all applications get automatically started at OS launch time, if this is available on the target OS. *This is also not supported on Unix.*

Unix specific shortcut attributes

This extension was programmed by MARC EPPELMANN. This is still in development and may be changed in one of the next releases of IzPack.

type - required

This must be one of **Application** or **Link**

- **Application**: To start any application, native, Java or shell-script based, the **type** has to be **Application**. The GUI-System will launch this Application, so as is, thru their native shell or application launcher. In this case, note that the right **workingDirectory** is always important on Unix platforms. If the users PATH environment variable does not contain the path, where the application is located, this can never be run, until the **workingDirectory** does not contain these path. The needed current path: ".", this is the case on most systems, should be in the users PATH environment variable. Consult the Unix manuals for more details.
- **Link**: If you want to open a URL in the users default Webbrowser, you have to set the **type** to **Link**. Note: The **url** attribute must be set to work properly.
- **Other**: There are more supported types on KDE, like **FSDevice**, but these types makes no sense for IzPack, in my opinion.

Without the type the Unix shortcut does not work.

url - semi-optional

If you want to create a shortcut as type *Link*, then you have to set the **url** attribute. The value can be a locally installed html or another document, with a known MIME type, like plain text, or a WWW Url i.e. 'http://www.izforge.com/izpack'.

A local document can be referenced by i.e. `"$INSTALL_PATH/doc/index.html"`.

The IzPack variable substitution system is supported by the **url**.

encoding - required

This should always set to **UTF-8**.

terminal - optional

If you want, the user can see the console output of a program (in Java applications `"System.outs"`), set the **terminal** attribute to **true**.

KdeSubstUID - unused

This is not fully implemented by IzPack. In the future this is the sudo option for a shortcut.

4.1.6 Selective Creation of Shortcuts

Usually all shortcuts that are listed will be created when the user clicks the 'Next' button. However it is possible to control to some degree if specific shortcuts should be created or not. This is based on install conditions. By including one or more `<createForPack name=''a pack name'' />` tags in the specification for a shortcut, you can direct the ShortcutPanel to create the shortcut only if any of the listed packs are actually installed. The 'name' attribute is used to define the name of one of the packs for which the shortcut should be created. You do not need to list all packs if a shortcut should always be created. In this case simply omit this tag altogether.

A word of caution

For any shortcut that is always created, I would recommend to omit this tag, since I have seen a number of problems related to changing pack names. You can save yourself some troubleshooting and some Aspirin by not using this feature if it's not required. On the other hand if you need it I would advise to be very careful about changing pack names.

4.1.7 Summary

Native Libraries

- ShellLink.dll - required by Microsoft Windows
- 'Nothing' - for KDE/Gnome shortcuts

Names of the Specification Files

shortcutSpec.xml for Windows and as default.

Unix_shortcutSpec.xml for Unix.

Specification File Layout - Windows

```
<shortcuts>
  <skipIfNotSupported/>
  <programGroup defaultName="MyOrganization\MyApplication"
    location="applications|startMenu"/>
  <shortcut
    name="Start MyApplication"
    target="$INSTALL_PATH\Path\to\MyApplication\launcher.bat"
    commandLine=""
    workingDirectory="$INSTALL_PATH\Path\to\MyApplication"
    description="This starts MyApplication"
    iconFile="$INSTALL_PATH\Path\to\MyApplication\Icons\start.ico"
    iconIndex="0"
    initialState="noShow||normal||maximized|minimized"
    programGroup="yes|no"
    desktop="yes|no"
    applications="yes|no"
    startMenu="yes|no"
    startup="yes|no">

    <createForPack name="MyApplication Binaries"/>
    <createForPack name="MyApplication Batchfiles"/>
  </shortcut>
</shortcuts>
```

A sample Specification File for Unix is at the end of this chapter

4.2 Shortcut Tips

I wrote this section to provide additional information about issues surrounding the creation of shortcuts. Reading this section is not necessary to successfully create shortcuts, but it might help you creating an installation that

works more smoothly. In addition, it might give you some knowledge about operating systems that you don't know so well. In fact most of the issues described in this section are focused on differences in operating system specifics.

4.2.1 The Desktop

You should recognize that the desktop is precious real estate for many people. They like to keep it uncluttered and keep only the things there that they use on a regular basis. This is not true for everybody and you might personally think different about this. Still, the fact remains that a lot of people might have different feelings about it, so you should not automatically assume that it is ok to place all of your shortcuts on the desktop proper. While your application is certainly one of the most important things for you, for your customers it is probably one of many applications they use and maybe not even the most important one. Accordingly, placing more shortcut icons there than they feel they will use on a regular basis and especially doing this without asking for permission might trigger some bad temper.

Annotation: But even the experienced user should be able to organize their Desktop. On Linux the users desktop is the only place, which supports any kind of shortcuts.

It is common practice to create a program group in the application menu system of the OS and place all shortcuts that go with an application in that program group. In addition, only one shortcut to the key access point of the application is placed directly on the desktop. Many installers first ask for permission to do so, as does the `ShortcutPanel` in `IzPack`.

I would like to recommend that you always create a shortcut in the menu system, even if your application has only one access point and you are placing this on the desktop. Note that shortcuts can also be placed directly in the menu, they don't need to be in a program group. There are two reasons for doing so.

- If the user elects not to create shortcuts on the desktop, they will end up with no access point to your application
- Even if this works fine, occasionally people 'clean up' their desktop. They might later find that they accidentally deleted the only access

point to your application. For the less technology savvy users, recreating the shortcut might be a rough experience.

4.2.2 Icons

Icons are supplied in image files, usually in some kind of bitmap format. Unfortunately there is no format that is universally recognized by all operating systems. If you would like to create shortcuts on a variety of operating systems that use your own icons, you must supply each icon in a number of different formats. This chapter discusses icon file formats used on various operating systems. Fortunately there are good programs available that allow you to convert between these formats, so that creating the different files is not much of a problem once the icons themselves are created.

Microsoft Windows

Windows prefers to use its native icon file format. Files of this type usually use the extension *.ico. These so called ICO files can hold multiple icons in one file, which can be useful if the same icon is to be provided in a number of sizes and color-depths.

Windows itself selects the icon with the most matching dimensions and displays it. While the Start menu displays the icon with 16x16 pixel if available, the desktop displays the 32x32 pixel resolution of the same ICO if this is in.

In other words, a ICO file has embedded one or more dimensions of the same Icon. We recommend to play with [microangelo](#).

Dlls and Exe files on the other side, can store, amongst other things, a collection of different Icons. You can select your desired Icon by its index. The lowest index is 0. Use the iconIndex attribute in the spec file to specify this index.

As a sample look into

```
%SystemRoot%\system32\shell32.dll
```

These contains a lot of Windows own icons. You can use the [PE Explorer](#) or another Resource Editor to extract or modify Icons in dlls or exe files. But be warned. You can also destroy a working application with these kind of tools.

At least Windows also supports the use of bitmap files in the *.bmp format as icons. Note that this format does not support multiple icons.

We might have overlooked other file formats that are supported by Windows. However, we suggest to test other formats for compatibility as they might not work all the way back to Windows 95 or on the NT/non-NT strain. Sticking with one of these two formats should keep you out of trouble.

Apple

Apple Macintosh systems use the Macintosh PICT format, extension *.pct. If you are working with an apple system you know a whole lot more about this format than I do. If you don't but would like to be able to install your application on a Mac, simply start with any bitmap format that you feel comfortable to work with. Then find an application that is capable of converting this format into a *.pct file. I like to use Paint Shop Pro (PC based), because it provides conversion capabilities among several dozen different file formats.

UNIX flavors

by Marc Eppelmann

As my knowledge, all X based Unix Window systems supports the (ASCII-) XBM (X-Bitmap) and the better XPM (X-Pixmap) format. The modern GUI systems like KDE and Gnome can display additionally a lot of other ImageIcon formats, such as GIF, JPG, and PNG.

I suggest to use PNG, because this can lossless compress like the GIF format, however this format is absolutely free. And not least, this can store true transparency informations (It has an alpha channel).

4.2.3 Targets

So, you thought you could escape the ugly mess of operating system dependencies at least with the way how your Java application is started? Sorry but I have just another bad message. The one positive thing is that here you have a way of escaping, even if doing so has a few less pretty side effects. At first, I would like to discuss various launching options you have available on different operating systems. At the end of the chapter I write about a way to make launching your application OS independent.

Microsoft Windows

On Microsoft Windows you have a variety of options for launching your application. Probably the most simple case is directly starting the Java VM from the command line and typing out all parameters, such as class path, the class name etc. In principle, this can be placed right in a shortcut and should work.

A little more elegant solution is to place this in a batch file and have the shortcut point to this batch file. This will also make it more likely that users can repair or recreate shortcuts. Recreating shortcuts with sophisticated command lines is practically impossible.

Another method is less commonly used but just as possible. Implement a native executable that launches the VM with your Java application. The VM comes as DLL and is used by java.exe in just the same way. As a sample look at the `exlipse.exe` provided by the [Eclipse-IDE](#)

Clearly, even though the first option is a bit ugly and has some restrictions, it is the most portable solution among the three.

Apple

We hope, there is a IzPack developer currently researching for the details for the Mac environment. We expect an updated chapter in one of the next releases.

UNIX

UNIX provides essentially the same options as Windows. You can simply use the command line option, you can write a shell script and you can write a native launcher. Naturally this stuff is in no way compatible with the equivalent Windows implementations. The native option is even more problematic in this environment, since the code can not even be moved from one UNIX platform to another, without recompilation.

OS Independent Launching

So, after all this rather discouraging news, there is actually a portable way to launch Java applications? You bet! although I have to admit that it is not necessarily the most pretty way of doing things.

This approach is currently used by IzPack. Package your application in a

*.jar file if you don't already do so and make it executable by including the necessary META-INF/MANIFEST.MF information file. I am not going into all the details on how exactly to do this, the Java documentation will have to do. You might have noticed that even though the instructions to install IzPack say to type :

```
java -jar IzPack-install.jar
```

You can just as well double click on IzPack-install.jar and it will start up. This procedure will work on all GUI based Java supported operating systems -though you might have to replace double clicking with dropping the file on the VM. In just the same way, you can make the *.jar file itself the target of a shortcut. Note: This works only, if jars are registered as files, which have to launch by the installed JRE (with: javaw.exe -jar *)

The one restriction with this approach is that a *.jar file can only have one main file. So, if you have multiple targets, they need to be packaged each into a different *.jar file. They can be in one *.jar file but then you have to start them explicitly, which gets you back to the problems that I mentioned before. This brings me to the ugly part. If you have just one target, then you are all set. If you have multiple targets, you need to create a *.jar file for each of them. In addition, you have a much harder time setting the classpath, because each of the *.jar files that contain supporting code must be listed. In fact, at present there is no way of setting this during the installation, because IzPack does not yet (version 3.0) support the setting and modification of environment variables.

4.2.4 Command Line

Before I start to write a lot about the use of command line arguments let me state this: If you can avoid using them, do it! Not that there is anything wrong with command line arguments as such. The issue is simply that if you want your application to be usable cross platform (the big Java promise) you should shy away from using command line arguments. The problem here is that not all operating systems actually support command line arguments. To be more precise, to my knowledge only Apple operating systems do not support command line parameters. If you don't care for running your application on a Mac, then you might not worry about this at all. If you are interested to support the Mac as well, read on.

In fact the Mac lower than MacOSX supports command line parameters in a way. More to the point, it supports a single parameter that your application should interpret as the name of a data file to open. You have no way of supplying this to your application through the command line attribute. The operating system generates this when the user drops the file on your application and then passes it as command line argument. That's it. This same behavior will probably fly well on pretty much any system and should therefore be an ok implementation.

So what to do if you want to modify program behavior based on runtime switches? For one thing, you could set system properties accordingly. The disadvantage here is the same as with the command line parameters: the way of setting these might vary between operating systems. The best way seems to be using a property file that contains the configuration data.

4.3 Trouble Shooting

by Elmar

It has been some time since I wrote this chapter during which a good number of users had a chance to gather experience. Unfortunately I never know how many have used it successfully without much difficulty. I only hear from those that have encountered one problem or another. The type of problems that I have seen prompted me to write this section, because I think it will help you in locating most problems that you might encounter or at least give you some idea where the problem might be located.

4.3.1 Problems You Can Solve

If you see an exception that essentially says that a library can not be loaded (ShellLink.dll) you have an easy problem to deal with. Your installer file is probably missing the native tag that adds the Windows dll to the installer or something with this tag is no quite right. Read 'What to Add to the Installer' for all details on this topic.

Most other problems cause the ShortcutPanel not to show at all during the installation process. The reason is simply that the ShortcutPanel skips if it does not know what to do or if it has nothing to do (no point showing then and confusing the user). The problem is that this is not always what you

intended. The most simple but not so uncommon case is, that the Shortcut-Panel cannot find their spec file. This can be caused by a number of reasons. The associated resource tag might be missing in the installer specification file, the target file name might be misspelled (the name you specify for the `id` attribute) or the target file name has a path or package name pre-pended. You have only to use `shortcutSpec.xml` or `Unix_shortcutSpec.xml` and nothing else, just as described in 'What to Add to the Installer'. You can always verify if this part is ok by inspecting the content of the installer *.jar file. The file `shortcutSpec.xml` should be located in the directory `res`. This inspection can be performed with any zip tool. If the file is not there, first correct this before proceeding.

If the file is there and the panel does not appear, you have a problem within the specification file. In most cases that I have seen, it comes down to a spelling mistake of an attribute or tag name. You just have to carefully make sure that everything is spelled correctly. Don't forget that all names are case sensitive! In a few cases it is also happened, that required or semi-optional attributes are omitted, so you might want to verify if all attributes that you need are actually supplied.

If everything is correct up to this point the problem becomes more elusive. Most likely the panel will not be displayed, because it is instructed not to show. There are be several reasons for this. The simple case is that no location has been specified for the shortcuts in your installation. This can happen if all five location attributes are omitted or if all the ones that are listed are set to `no`. Remember, you have to specify at least one location for every shortcut. If this is also correct, you might have used the `<createForPack>` tag. Review the details in 'Selective Creation of Shortcuts'. One possibility for the panel not to show is that based on the packs that are currently selected for installation no shortcut qualifies for creation. In this case the panel will not show, this is perfectly normal behavior. More likely this condition is true because of some accident and not because it's intended. Make sure the packs that you list for the shortcut are actually defined in your installation and verify that they are all spelled correctly. Remember: case matters! Did the ShortcutPanel use to work in your installation and all of a sudden stopped working? Very likely you are dealing with the last problem. A package name might have been modified and the shortcut spec was not adjusted to stay in sync.

4.3.2 Problems That Have No Solution (yet)

Unfortunately one problem has been very persistent and only recently one user found the reason. The problem occurs when installing on some target systems where non-English characters are used in the storage path for the shortcuts. The problem is that these characters don't seem to be properly translated across the Java Native Interface. This leads to a situation where the proper path can not be located and the shortcut creation fails. I write 'some target systems' because it does not fail everywhere. After much agonizing over this problem, one user found the solution: The shortcut creation works fine if a Sun virtual machine is installed, but fails if a version from IBM happens to be installed. So far I have no solution for this problem but I am trying to find a workaround the problem.

4.3.3 A sample shortcut specification file for Unix

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<shortcuts>

  <programGroup defaultName="IzForge/IzPack" location="applications"/>

    <!-- Disabled since there is no Frontend
shortcut
  name="IzPack"
  programGroup="yes"
  desktop="yes"
  applications="no"
  startMenu="yes"
  startup="no"
  target="$INSTALL_PATH/bin/izpack-fe.sh"
  commandLine=""
  workingDirectory="$INSTALL_PATH/bin"
  description="Front-End for IzPack installation tool"
  iconFile="$INSTALL_PATH/bin/icons/izpack.png"
  iconIndex="0"
  type="Application"
  encoding="UTF-8"
  terminal="true"
  KdeSubstUID="false"
  initialState="normal">
    <createForPack name="Core"/>
  </shortcut -->

</shortcut
```



```

    name="IzPack Documentation"
    programGroup="yes"
    desktop="yes"
    applications="no"
    startMenu="yes"
    startup="no"
    target="konqueror"
    workingDirectory=""
    commandLine=""
    initialState="noShow"
    iconFile="help"
    iconIndex="0"
    url="$INSTALL_PATH/doc/izpack/html/izpack-doc.html"
    type="Link"
    encoding="UTF-8"
    description="IzPack user documentation (HTML format)">

    <createForPack name="Documentation-HTML"/>
</shortcut>

<shortcut
    name="Documentation"
    programGroup="yes"
    desktop="yes"
    applications="no"
    startMenu="yes"
    startup="no"
    target="acroread"
    workingDirectory=""
    commandLine="$INSTALL_PATH/doc/izpack/pdf/izpack-doc.pdf"
    initialState="noShow"
    iconFile="acroread"
    iconIndex="0"
    url="$INSTALL_PATH/doc/izpack/pdf/izpack-doc.pdf"
    type="Application"
    encoding="UTF-8"
    description="IzPack user documentation (PDF format)">

    <createForPack name="Documentation-PDF"/>
</shortcut>

<shortcut
    name="Uninstaller"
    programGroup="yes"
    desktop="yes"
    applications="no"
    startMenu="no"
    startup="no"
    target="/usr/lib/java/bin/java"

```

```
commandLine="-jar &quot;${INSTALL_PATH}/Uninstaller/uninstaller.jar&quot;;"  
initialState="noShow"  
iconFile="trashcan_full"  
iconIndex="0"  
workingDirectory=""  
type="Application"  
encoding="UTF-8"  
description="IzPack uninstaller">  
  <createForPack name="Core" />  
</shortcut>  
</shortcuts>
```

Chapter 5

Creating Your Own Panels

5.1 How It Works

5.1.1 What You Need

First you have to read the NanoXML documentation if you need to use XML in your panel. Secondly, it is necessary that you use the Javadoc-generated class references. We will just explain here briefly how to start making your panels.

It is a good idea to read the source code of some IzPack panels. They are usually very small, which makes it easier to understand how to write your own.

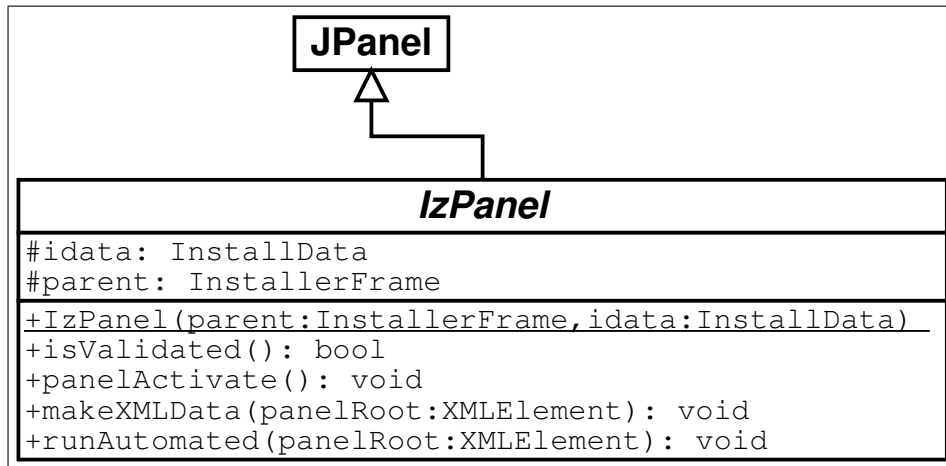
5.1.2 What You Have To Do

Extending IZPACK with a panel is quite simple. A panel used with IZPACK must be a subclass of `IzPanel`. The `IzPanel` class is located in the `com.izforge.izpack.installer` package but your panels need to belong to `com.izforge.izpack.panels`.

Things will get a good deal easier if you build IzPack with Jakarta Ant. Simply add your class in the source tree and add the `And` directives to build your own panels. In this way you'll be able to focus on your code :-)

5.2 The IzPanel Class

5.2.1 UML Diagram



5.2.2 Description

The data members are : the install data (refer to the **InstallData** Javadoc reference) and a reference to the parent installer frame. Additional there are the initialFocus Component and some members for handling the grid bag constraint.

The methods have the following functionality :

- (*constructor*) : called just after the language selection dialog. All the panels are constructed at this time and then the installer is shown. So be aware of the fact that the installer window is **not** yet visible when the panel is created. If you need to do some work when the window is created, it is in most cases better to do it in **panelActivate**.
- **isValidated** returns **true** if the user is allowed to go a step further in the installation process. Returning **false** will lock it. For instance the **LicencePanel** returns **true** only if the user has agreed with the license agreement. The default is to return **true**.

- **panelActivate** is called when the panel becomes active. This is the best place for most initialization tasks. The default is to do nothing.
- **makeXMLData** is called to build the automated installer data. The default is to do nothing. **panelRoot** refers to the node in the XML tree where you can save your data. Each panel is given a node. You can organize it as you want with the markups you want starting from **panelRoot**. It's that simple.
- **runAutomated** is called by an automated-mode installation. Each panel is called and can do its job by picking the data collected during a previous installation as saved in **panelRoot** by **makeXMLData**.
- **setInitialFocus** with this method it is possible to set a hint which component should be get the focus at activation of the panel. It is only a hint. Not all components are supported. For more information see `java.awt.Component.requestFocusInWindow` or `java.awt.Component.requestFocus` if the VM version is less than 1.4.
- **getInitialFocus** returns the component which should be get the focus at activation of the panel. If no component was set, null returns.

Additional there are some helper methods to simplify grid bag layout handling and creation of some common used components.

Chapter 6

User Input

(by Elmar GROM)

Most of the panels that come with IzPack take user input in some form. In some panels this is through a simple user acknowledgment in others the user can enter text or select a directory through a file open dialog. In all of those cases the user input is used for the specific purpose needed by the panel that takes the input. However, if you need user input during installation that will later on be available to your application then you need to use the user input panel.

To use this panel, list it in the install file with the class name `UserInputPanel`. In addition, you must write a XML specification and add it to the install resources. The name of this resource must be `userInputSpec.xml`.

The user input panel is a blank panel that can be populated with UI elements through a XML specification file. The specification supports text labels, input elements, explanatory text and some minor formatting options.

The following types of user input elements are supported:

- Text
- Combo Box
- Radio Buttons
- Check Box
- Rule Input Field

- Search Field

The way in which this panel conveys the user input to your application is through the variable substitution system. User input is not directly inserted into your configuration files but the variables that you specify for this panel are set in the variable substitution system. After this operation has taken place the variables and associated values are available for all substitutions made. This way of operation has a number of implications that you should be aware of.

First, not only can you set additional variables in this way but you can also modify variables that are defined elsewhere -even built in variables. For this reason you should be careful to avoid overlaps when choosing variable names. Although there might be cases when it seems useful to modify the value of other variables, it is generally not a good idea to do so. Because you might not exactly know when other variables are set and when and where they are used throughout the installation process, there might be unintended side effects.

Second, the panel must be shown at a point during the installation process before the variables are used. In most cases you will use the values to substitute variables in launch and configuration files that you supply with your installation. For this to work you place this panel before the install panel, because the install panel uses the variable substitutor to replace all such variables. Although using this panel any later in the process will correctly set the variables internally, there won't be any affect on the files written to disk. You can also use variables set in this way in other panels that you have written yourself. There is a section in the chapter on writing your own panel that explains how to do this. Also in this case it is important to place the associated input panel in the process before the variables are used.

At this point I would also like to mention that it is possible to hide select elements on the panel or the panel altogether if certain packs are not selected. For this to work you must place this panel after the packs panel. One side effect of using this feature is that it is not possible to step back once the user input panel is displayed. This is because the user might make changes in the packs selection that would require a complete rebuild of the UI. Unfortunately, building the UI is an irreversible process, therefore the user can not be allowed to go back to the packs panel.

6.1 The Basic XML Structure

The top level XML section is called `<userInput>`. For most panels it does not make sense to present them more than once, however you might want to present multiple user input panels -with different content of course. Therefore the `<userInput>` section can contain multiple tags that each specify the details for one panel instance. The tag name for this is `<panel>`.

The `<panel>` tag uses the following attributes:

order - required

This is the order number of the user input panel for which this specification should be used. Counting starts at 0 and increments by 1 for each instance of the user input panel. So if a spec should be used for the second occurrence of the user input panel use `order="1"`.

layout - optional

There are three general layout rules this panel uses, they are `left`, `center` and `right`. While I think left is most commonly used, you might want to experiment with this attribute and see which you like best. The default is `left`.

6.2 Concepts and XML Elements Common to All Fields

Before I dive into the details of defining the various UI elements I would like to present XML elements and general concepts that apply throughout. This saves me a lot of work in writing and you a lot of repetitive reading and maybe a tree or two.

The UI elements are generally laid out top to bottom in the order they appear in the XML file. The only exception to this rule is the title, which always appears at the very top. The layout pattern for the input fields is as follows: If a description is defined, it appears first, using the full available layout width. The input field is placed beneath the description. With fields such as the text field or the combo box, the label is placed to the left and the input field to the right. Fields such as radio buttons and check boxes are

somewhat indented and have the label text appear to their right.

Each UI element is specified with a `<field>` tag. The `type` attribute is used to specify what kind of field you want to place. Obviously, the `type` attribute is not optional.

Each field that takes user input must also specify the variable that should be substituted. This is done with the `variable` attribute.

Almost all fields allow a description. When a description is allowed it is always added in the same way. The description is part of the data within the field tag. There can only be one description per field. If you add more than one, the first one is used and the others ignored. There are three attributes used with this tag. The text is specified through the `txt` or the `id` attribute. The details on using them are described below. The attributes are all optional but you must specify text to use, either directly or through the `id` attribute. In addition, you can set the text justification to `left`, `center` and `right` with the `align` attribute.

The following example illustrates the general pattern for field specification:

```
<field type="text" variable="myFirstVariable">
  <description align="left" txt="A description" id="description 1"/>
  .
  .
  .
</field>
```

A very frequently used pattern is for the definition of text. Where ever text is needed (labels, descriptions, static text, choices etc.) it can be specified in place using the `txt` attribute. This is convenient if you are only supporting a single language. However, if you would like to separate your text definitions from the panel specification or if you need to support multiple languages you might want to use the `id` attribute instead to only specify an identifier. You can then add multiple XML files with the same name as this spec file (userInputSpec.xml) appended with an underscore '_' and the appropriate three letter ISO3 language code. The content of those files must conform to the specification for IzPack language packages. For more details on this topic see the chapter on language packages under advanced features. `id` defines an identifier that is also defined in the language package,

together with the localized text to use. It is possible to use both the `txt` and the `id` attribute. In this case the text from the language package is used. If for some reason the language package is not available or the `id` is not defined there, the text specified with `txt` is used as default.

All input fields can be pre-set with a value of your choice. Although the details vary a bit from field type to field type, the `set` attribute is always used to accomplish this. The `set` attribute is of course optional.

All fields that take user input use a `<spec>` tag to define the details of the input field. In some cases the content of this tag is rather simple. Input fields with a more complex nature tend to have accordingly complex content in this tag. Since the details vary widely, they are explained with each input field.

Any number of `<createForPack name='a pack name' />` tags can be added to the `<panel>` and `<field>` sections. This tag has only one attribute and no data. The attribute is `name` and specifies the name of one of the installation packs that you have defined. Here is how it works: if no `<createForPack ...>` tag exists in a section, the entity is always created. However, if the tag exists, the entity is only created if one or more of the listed packs are selected for installation. As mentioned before, if you are using this feature, make sure the user input panel shows up after the packs panel.

6.3 Internationalization

To provide internationalization you can create a file named `userInputLang.xml_xyz` where `xyz` is the ISO3 code of the language in lowercase. Please be aware that case is significant. This file has to be inserted in the resources section of `install.xml` with the `id` and `src` attributes set at the name of the file.

Example:

If you have the following `userInputSpec.xml` and you want to internationalize `input.comment`, `input.proxy`, `input.port` for english and french you have to create two files named `userInputLang.xml_eng` and `userInputLang.xml_fra`:

```
<userInput>
```

```

<panel order="0">
  <field type="staticText" align="left" txt="My comment is here." id="input.comment">
  <field type="text" variable="proxyAddress">
    <spec txt="Proxy Host:" id="input.proxy" size="25" set=""/>
  </field>
  <field type="text" variable="proxyPort">
    <spec txt="Proxy Port:" id="input.port" size="6" set=""/>
  </field>
</panel>
</userInput>

```

userInputLang.xml_eng file contains:

```

<langpack>
  <str id="input.comment" txt="English:My comment is here."/>
  <str id="input.proxy" txt="English:Proxy Host:"/>
  <str id="input.port" txt="English:Proxy Port:"/>
</langpack>

```

userInputLang.xml_fra file contains:

```

<langpack>
  <str id="input.comment" txt="French:My comment is here."/>
  <str id="input.proxy" txt="French:Proxy Host:"/>
  <str id="input.port" txt="French:Proxy Port:"/>
</langpack>

```

you will also have to add the following to the install.xml file

```

<resources>
  ...
  <res id="userInputSpec.xml" src="userInputSpec.xml"/>
  <res id="userInputLang.xml_eng" src="userInputLang.xml_eng" />
  <res id="userInputLang.xml_fra" src="userInputLang.xml_fra" />
  ...
</resources>

```

6.4 Panel Title

You can place an optional title at the top of the panel. Though it is not possible to select a font for the title that is different from the one used on the rest of the panel, it is possible to modify the font to some extent. To specify the

title create a `<field>` tag and use the `type` attribute with the value `title`. In addition to the `txt` and `id` attributes, the following attributes are supported:

italic - optional

With a value of `true` specifies that the title font should be in italics.

bold - optional

With a value of `true` specifies that the title font should be bold.

size - optional

This attribute specifies the size of the title font. Please note that the size is not specified in points but as a relative size multiplier compared to the body font on the panel. The default value is 2.

6.5 Static Text

Static text is simply text that is placed on the panel without direct connection to any of the input elements. It is laid out to use the entire layout width available on the panel and is broken into multiple lines if necessary. To specify static text create a `<field>` tag and use the `type` attribute with a value of `staticText`. In addition to the `txt` and `id` attributes, the text can be justified `left`, `center` or `right` with the `align` attribute. It is not possible to format this text in any way.

Example

The following example inserts some static text in the panel.

```
<field type="staticText" align="left"
      txt="This is just some simple static text."
      id="staticText.text"/>
```

6.6 Visual Separation

Sometimes it is desirable to separate different entities visually. This can be accomplished by inserting a space or a divider. A space simply inserts a

vertical separation of the average height of a single line entity, such as a line of text or a an input field. A divider inserts the same amount of space but also draws a division line which can be either aligned at the top or bottom of the separation. `<space>`, `<divider>`

..... maybe I should draw the line myself and add no additional space at all ...

6.7 Text Input

A text input field allows the user to enter and edit a single line of text, without length restriction. The input field can have a label, which will show to the left of the input field and a description, which can span multiple lines. The description is placed above the input field and uses the entire available layout width. The width of the input field must be explicitly set, otherwise it will only accommodate a single character. To specify a text input field create a `<field>` tag and use the `type` attribute with a value of `text`. The `txt` and `id` attributes are not supported here. The `variable` attribute specifies the variable that should be replaced with the text taken from the input field.

The Data

The data consists of two items, a description and the spec. The `<spec>` tag uses four attributes. The label text is specified with `txt` and/or `id` as described above. In addition, the width of the input field as it appears on the panel can be set with the `size` attribute. The value must be an integer and sets the field width based on the average character width of the active font. If this is not specified, then you will end up with a very narrow field that is practically unusable.

The fourth attribute `set` is optional. It takes a text string to pre-fill the input field.

Example

The following example creates a text input field with a label and description. The width of the input field will be enough to accommodate 15 characters. The field will be pre-set with the text 'some text' when the UI is first presented.

```
<field type="text" variable="textInput">
```

```

<description align="left" txt="A description for a text input field"
            id="description.text"/>
<spec txt="Enter some text:" id="text.label" size="15" set="some text"/>
</field>

```

6.8 Radio Buttons

The radio buttons are useful when the user needs to select a specific option out of a pre-defined list of choices. This field offers an arbitrary number of mutually exclusive buttons, each with its own label. The placement of the buttons and labels is different from other fields. First, the button is placed to the left and the label text to the right. Second, the buttons are not lined up all the way to the left as other labels are but they are indented from that location. As with other fields, the description is placed above the list of radio buttons and uses the entire available layout width. To specify a set of radio buttons create a `<field>` tag and use the `type` attribute with a value of `radio`. The `txt` and `id` attributes are **not** supported here. As with all other input fields, the `variable` attribute specifies that variable that should be replaced with the user selection.

The Data

The data consists of two items, a description and the spec. The `<spec>` tag has no attributes, instead the specification details are entered as data within the `<spec>` tag. The `<spec>` data consists of one or more `<choice>` tags. One `<choice>` tag is required for each radio button. The `<choice>` tag accepts the usual `txt` and `id` attributes, which are used to specify the label text. In addition the following attributes are supported:

value - required

The `value` attribute is used to specify which value to insert if this associated radio button is selected. In other words, the label text has nothing to do with the value that is actually substituted for the variable. For this reason there is never an issue if multiple languages are used, the value is always the same for a given selection.

set - optional

The `set` attribute accepts the values `true` and `false`. Since the attribute is optional it can also be omitted, which is interpreted as `false`. If a value

of `true` is used, the associated radio button will be selected when the UI is first presented. Obviously, only one of the buttons in a set should be set to `true`.

Example

The following example creates a set of four radio buttons with description. The second button will be selected when the UI is first presented.

```
<field type="radio" variable="radioSelection">
  <description align="left" txt="This is a description for radio buttons"
    id="description.radio"/>
  <spec>
    <choice txt="the first choice" id="radio.label.1" value="1 selected" />
    <choice txt="the second choice" id="radio.label.2" value="2 selected"
      set="true" />
    <choice txt="the third choice" id="radio.label.3" value="3 selected" />
    <choice txt="the fourth choice" id="radio.label.4" value="4 selected" />
  </spec>
</field>
```

6.9 Combo Box

The combo box provides essentially the same functionality as do the radio buttons, just in a different presentation style. The advantage of the combo box is that it is easier to deal with a long list of choices.

6.10 Check Box

If there are a number of choices and any combination of them could be selected, not just a single one, then radio buttons are not the way to go. You might be better off using a number of check boxes. The layout for a check box works in the same way as for radio buttons. The check box is placed indented from the left most edge and the label text is placed to the right of it. Other than with radio buttons, you cannot define any number of check boxes. This field allows the definition of only one check box, which is associated with one variable. If you need multiple check boxes you need to define one field for each of them. To make it look like a cohesive group you simply provide a description only for the first check box. All of the check boxes will be positioned in such a way that they look like a group, even though they

are separate entities and their selections are conveyed to different variables. The description is placed above the check box and uses the entire available layout width. To specify a check box create a `<field>` tag and use the `type` attribute with a value of `check`. As with all other input fields, the `variable` attribute specifies the variable that should be replaced with the user input.

The Data

The data consists of two items, a description and the spec. The `<spec>` tag accepts the usual `txt` and `id` attributes, which are used to specify the label text. In addition, the following attributes are supported:

`true` - required

The `true` attribute specifies the value to use for substitution when the box is selected.

`false` - required

The `false` attribute specifies the value to use for substitution when the box is not selected.

`set` - optional

The `set` attribute accepts the values `true` and `false`. Since the attribute is optional it can also be omitted, which is interpreted as `false`. If a value of `true` is used, the check box will be selected when the UI is first presented.

Example

The following example creates a check box with description. The check box will not be selected when the UI is first presented. This could also be accomplished by omitting the `set` attribute.

```
<field type="check" variable="chekSelection.1">
  <description align="left" txt="This is a description for a check box"
    id="description.check.1"/>
  <spec txt="check box 1" id="check.label.1" true="on" false="off"
    set="false"/>
</field>
```


6.11 Rule Input

The rule input field is the most powerful and complex one of all the input fields offered by this panel. In its most simple incarnation it looks and works like a regular text input field. There is also only an incremental increase of the complexity in the specification for this case. However, it is unlikely that you would use it for such a purpose. The real power of this input field comes from the fact that rules can be applied to it that control many aspects of its look as well as overt and covert operation.

6.11.1 Layout and Input Rules

The basic nature of this input field is that of a text input field and as mentioned before, in its most simple incarnation that is what it looks like and how it operates. However, the layout of the field can be defined in such a way that there are multiple logically interconnected text input fields, adorned with multiple labels. Further more, each of these fields can be instructed to restrict the type of input that will be accepted. Now you might ask what this could be useful for. As an answer, let me present a few examples that show how this feature can be used. Before I do this however, I would like to describe the specification syntax, so that the examples can be presented together with the specifications that make them work in a meaningful way.

The actual specification of the layout, the labels and the type of input each field accepts all happens in a single string with the `layout` attribute. First let us have a look at the specification format for a single field. This format consists of a triplet of information, separated by two colons `':'`. A typical field spec would look like this: `N:4:4`, where the first item is a key that specifies the type of input this particular field will accept - numeric input in the example. The second item is an integer number that specifies the physical width of the field, this is the same as in the width of any regular text field. Therefore the field in the example will provide space to display four characters. The third item specifies the editing length of the string or in other words, the maximum length of the string that will be accepted by the field. In the `layout` string you can list as many fields as you need, each with its own set of limitations. In addition you can add text at the front, the end and in between the fields. The various entities must be separated by white space. The behavior of this field is such that when the editing length of a field has been reached, the cursor automatically moves on to the next field. Also, when the backspace key is used to delete characters and the beginning

of a field has been reached, the cursor automatically moves on to the previous field. So let us have a look at some examples.

Phone Number

The following specification will produce a pre formatted input field to accept a US phone number with in-house extension. Even though the pattern is formatted into number groups as customary, complete with parentheses '(' and dash '-', entering the number is as simple as typing all the digits. There is no need to advance using the tab key or to enter formatting characters. Because the fields only allow numeric entry, there is a much reduced chance for entering erroneous information. "(N:3:3) N:3:3 - N:4:4 x N:5:5". Each of the fields uses the 'N' key, indicating that only numerals will be accepted. Also, each of the fields only accepts strings of the same length as the physical width of the field.

E-Mail Address

This specification creates a pattern that is useful for entering an e-mail address "AN:15:U @ AN:10:40 . A:4:4". Even though the first field is only fifteen characters wide it will accept a string of unlimited length, because the 'U' identifier is used for the edit length. The second field is a bit more restrictive by only accepting a string up to forty characters long.

IP Address

It might not be uncommon to require entering of an IP address. The following simple specification will produce the necessary input field. All fields are the same, allowing just three digits of numerical entry. "N:3:3 . N:3:3 . N:3:3 . N:3:3"

Serial Number or Key Code

If you ship your product with a CD key code or serial number and require this information for registration, you might want to ask the customer to transcribe that number from the CD label, so that it is later on accessible to your application. As this is always an error prone operation, the predefined pattern with the easy editing support and restriction of accepted data helps to reduce transcription errors "H:4:4 - N:6:6 - N:3:3". This particular specification will produce three fields, the first accepting four hexadecimal, the second six numerical and the third three numerical digits.



Limitations

Even though the above examples all use single character labels between fields, there is no restriction on the length of these labels. In addition, it is possible to place label text in front of the first field and after the last field and the text can even contain spaces. The only limitation in this regard is the fact that all white space in the text will be reduced to a single space on the display. This means that it is not possible to use multiple spaces or tabs in the text.

The following table lists and describes all the keys that can be used in the specification string.

<i>Key</i>	<i>Meaning</i>	<i>Description</i>
N	numeric	The field will accept only numerals.
H	hexadecimal	The field will accept only hexadecimal numerals, that is all numbers from 0-F.
A	alphabetic	The field will accept only alphabetic characters. Numerals and punctuation marks will not be accepted.
AN	alpha-numeric	The field will accept alphabetic characters and numerals but no punctuation marks.
O	open	The field will accept any input, without restriction.
U	unlimited	This key is only legal for specifying the editing length of a field. If used, the field imposes no length restriction on the text entered.

6.11.2 Setting Field Content

Like all other input fields the rule input field can also be pre-filled with data and as usual, this is accomplished through the **set** attribute. As you might expect, the details of setting this field are rather on the complicated side. In fact you can set each sub field individually and you can leave some of the fields blank in the process. The **set** specification for all sub fields is given in a single string. Each field is addressed by its index number, with the count starting at 0. The index is followed by a colon ':' and then by the content of the field. The string "0:1234 1:af415 3:awer" would fill the first subfield with 1234, the second one with af415 and the fourth with awer. The third subfield would stay blank and so would any additional fields that might follow.

The individual field specs must be separated with spaces. Spaces within the pre-fill values are not allowed, otherwise the result is undefined.

6.11.3 The Output Format

The user input from all subfields is combined into one single value and used to replace the variable associated with the field. You can make a number of choices when it comes to the way how the subfield content is combined. This is done with the **resultFormat** and **separator** attributes. The **resultFormat** attribute can take the following values:

<i>Value</i>	<i>Meaning</i>
plainString	The content of all subfields is simply concatenated into one long string.
displayFormat	The content of all subfields and all labels -as displayed- is concatenated into one long string.
specialSeparator	The content of all subfields is concatenated into one string, using the string specified with the separator attribute to separate the content of the subfields.
processed	The content is processed by Java code that you supply before replacing the variable. How to do this is described below.

6.11.4 Validating the Field Content

You can provide runtime validation for user input into a rule field via the `validator` element (which is a child of the `field` element). There are two types of built-in validators already provided: a `NotEmptyValidator` and a `RegularExpressionValidator`. You can also easily create your own validator. In all cases, if the chosen validator returns `false`, a messagebox will display the contents of the `txt` attribute and the user will be unable to continue to the next panel.

You can specify a processor for a combobox:

```
<choice processor="fully.qualified.class.name"
        set="selectedValue"/>
```

so that you can fill a combobox with data on a simple way.

NotEmptyValidator

The `NotEmptyValidator` simply checks that the user entered a non-null value into each subfield, and returns `false` otherwise.

RegularExpressionValidator

The `RegularExpressionValidator` checks that the user entered a value which matches a specified regular expression, as accepted by the Jakarta Regexp library (<http://jakarta.apache.org/regexp/>). The syntax of this implementation is described in the javadoc of the `RE` class (<http://jakarta.apache.org/regexp/apidocs/org/apache/regexp/RE.html>).

You can specify the regular expression to be tested by passing a parameter with a name of `pattern` to the validator (via the `param` element), with the regular expression as the `value` attribute. For example, the following would validate an e-mail address:

```
<field type="rule" variable="EMAILADDRESS">
  <spec
    txt="Your Email Address:" layout="0:12:U @ 0:8:40 . A:4:4"
    set="0: 1:domain 2:com" resultFormat="displayFormat"
  />
  <validator class="com.izforge.izpack.util.RegularExpressionValidator"
    txt="Invalid email address!">
    <param
```

```

        name="pattern"
        value="[a-zA-Z0-9._-]{3,}@[a-zA-Z0-9._-]+([\.[a-zA-Z0-9._-]+)*[\.[a-zA-Z0-9._-]{2,4}"
    />
</validator>
</field>

```

You can test your own regular expressions using the handy applet at <http://jakarta.apache.org/regexp/applet.html> .

Creation Your Own Custom Validator

You can create your own custom Validator implementation simply by creating a new class which implements the `com.izforge.izpack.panels.Validator` interface. This interface specifies a single method: `validate(ProcessingClient client)` , which returns a `boolean` value. You can retrieve the value entered by the user by casting the input `ProcessingClient` as a `RuleInputField` and calling the

`RuleInputField.getText()` method. You can also retrieve any parameters to your custom `Validator` by calling the

`RuleInputField.getValidatorParams()` which returns a `java.util.Map` object containing parameter names mapped to parameter values. For an example, take a look at

`com.izforge.izpack.util.RegularExpressionValidator`.

Set values in the `RuleInputField` can be preprocessed. At now you can specify a processor class to pre process a value to be set at initial value of a `RuleInputField`. Syntax:

```
<spec set="0:defaultVal:classname" .../>
```

The class name is an optional value. The class must implement the `Processor` interface.

6.11.5 Processing the Field Content

This feature needs to be documented.

6.11.6 Summary Example

```

<field type="rule" variable="test1">
    <description align="left" txt="A description for a rule input field."

```

```

        id="description.rule.1"/>
<spec txt="Please enter your phone number:"
      layout="( N:3:3 ) N:3:3 - N:4:4 x N:5:5"
      resultFormat="specialSeparator" separator="." />
<validator class="com.izforge.izpack.util.NotEmptyValidator"
      txt="The phone number is mandatory!" />
<!--processor class="" /-->
</field>

```

6.12 Search

The search input field allows the user to choose the location of files or directories. It also supports auto-detection of the location using a list of suggestions. The field is basically a combobox with an extra button to trigger auto-detection (again).



6.12.1 Specification

The `<description>` tag is the same as with other fields (see 6.2 on page 72). The `<spec>` tag supports the following attributes:

- **filename** - the name of the file or directory to search for
- **type** - what to search for
 - **file** - search for a file
 - **directory** - search for a directory
- **result** - what to return as the search result
 - **file** - result of search is whole pathname of file or directory found
 - **directory** - only return directory where the file was found (this is the same as **file** when searching for directories)
 - **parentdir** - return the full path of the parent directory where the file was found
- **checkfilename** - the name of a file or directory to check for existence (this can be used to validate the user's selection)

6.12.2 Example

```
<field type="search" variable="java_sdk_home">
  <description align="left"
    txt="This is a description for a search input field."
    id="description.java_sdk_home"/>
  <spec txt="Path to Java SDK:" checkfilename="lib/tools.jar"
    type="file" result="directory">
    <choice value="/usr/lib/java/" os="unix" />
    <choice value="/opt/java" os="unix" />
    <choice value="C:\Program Files\Java" os="windows" />
    <choice value="C:\Java" os="windows" />
  </spec>
</field>
```


Chapter 7

Custom Actions

(by Klaus BARTZ)

7.1 Overview

In general the installation procedure is separated into several steps. The first step, let's call it the *data collection phase*, is getting specific data needed for the installation process. Typically this is done by typing all needed data into one or more panels, if a GUI is used, or automatically by reading the data from a config file. In general nothing will be changed on the system until all needed data is obtained. But mostly - depending on to the information, e.g. the destination path - different input panels are involved.

If all needed data is collected the second step will be performed, let us call it the *action phase*. During this step the state of the locale machine will be changed, e.g. files will be copied to the installation destination or some short cuts will be registered. Each of this subsequent steps are denoted as actions. There are actions intended to be reused, so called common actions, and actions for one special purpose only, so called custom actions. In IzPack there are already some common actions, for example "file transfer", "parse" or "execute".

The third step, the *reporting phase*, is normally represented by a panel that reports the result state of the installation (OK, or not OK) and a simple good bye message.

With IzPack there are two ways to implement custom actions. Firstly it is always possible to define a custom panel that performs the desired actions too. Secondly, and that's the new, custom actions are supported.

Panels still may be used for actions that are performed, e.g. before files are transferred or after the "execute" action. But if the needed action de-

depends on the selected or already installed packages, this works also, but the implementation effort is much higher.

If the action should be performed for several amount of elements of a pack, using custom actions will be more easy than using panels. Additional custom actions may be defined for installation, but also for packaging and uninstallation purposes. If a custom action is also needed for uninstallation purposes, it'll be always a good idea to implement a corresponding installation action as custom action, but not as panel.

7.2 How It Works

Custom actions are implemented as listeners. Each listener implements callback methods that will be called at well-defined points. The method `InstallerListener.afterFile` for example will be called after a file has been copied. There are different interfaces intended for being used at packaging time, at installation time and at uninstallation time.

Each interface is implemented by a class with the prefix "Simple" (e.g. `SimpleCompilerListener`) that implements all declared interface methods with an empty body. These classes may be used as base classes for own listener implementations.

To apply custom actions to the installer, an entry in the appropriate `install.xml` file is needed. The configuration of listeners starts with the facultative `ELEMENT "listeners"` which can contain one or more `ELEMENTs` of `"listener"`. For a `"listener"` there are three attributes which determine the `"compiler"`, `"installer"` and `"uninstaller"` custom action pupose. Additionally it is possible to make the listener OS dependent using the `"os"` `ELEMENT`.

If file related data will be set, the facultative `ELEMENT "additionaldata"` is defined for the `ELEMENTs` `"file"`, `"singlefile"` and `"fileset"`. This data will be automatically moved to the corresponding `PackFile` objects in the `install.jar`. Extraction and usage should be implemented in a install custom action (see example).

7.2.1 Custom Action Types

Custom actions are intended to be used at packaging time, at installation time and at uninstallation time. The interfaces are:

<i>Custom action type</i>	<i>Interface name</i>
Packaging	<code>com.izforge.izpack.event.CompilerListener</code>
Installation	<code>com.izforge.izpack.event.InstallerListener</code>
Uninstallation	<code>com.izforge.izpack.event.UninstallerListener</code>

Custom Actions At Packaging

UML Diagram

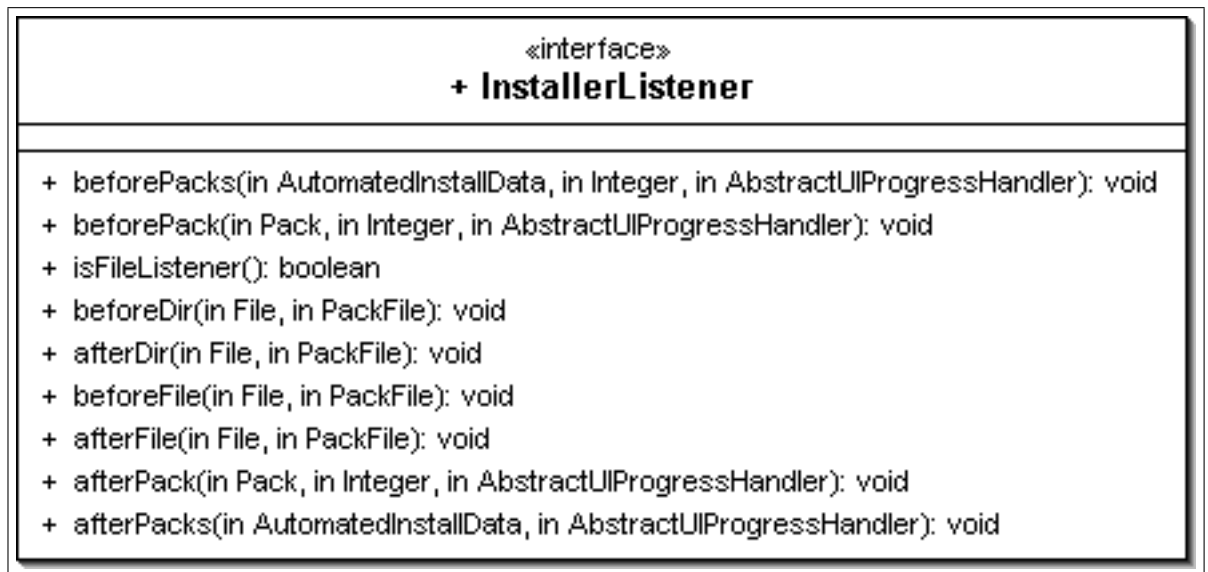


Description

- (*constructor*) : only the default constructor will be used. It is called from `Compiler` just after creating the packager. Therefore initializing will be better during in the first `notify` call.
- `reviseAdditionalDataMap` gives the facility to add data to each `PackFile` object. This is the place where file related data can be transferred from the install xml file into the install jar file. Although each key and value of the map can be any type, but the class definitions of all used types must therefore be contained in the installer jar file or in the VM's classpath. In general strings are the best choice for being used as keys or values. All keys must be unique over all registered `CompilerListeners`. Each call of this method adds own key value pairs to the given `existenDataMap` because more than one listener can be used. If the given map is null, a new one will be created.
- `notify` is called at the beginning and at the end of each "add" method call which is called in `Compiler.executeCompiler`.

Custom Actions At Installing Time

UML Diagram



Description

- (*constructor*) : only the default constructor will be used. It is called from `Unpacker.run` before unpacking.
- `beforePacks` will be called each time before an unpacking call is performed.
- `beforePack` is called before a package is installed. `Pack` object and the number of the pack are passed.
- `isFileListener` determines whether the next four methods are called or not. This is a little performance optimizing.
- `beforeDir` is called before a directory is created. In this case, when file listeners exist, directories are created recursively and the method is called at each step. The file and the current `PackFile` object are passed.
- `afterDir` is called directly after the directory creation.
- `beforeFile` is called before a file is created. The file and `PackFile` object are passed as parameters.

- `afterFile` is the best place to perform file related actions. The given `PackFile` objects contains the additional data which was set at packaging.
- `afterPack` will be just called after the pack is closed.
- `afterPacks` is the last step before the handler will be stopped.

Custom Actions At Uninstalling Time

UML Diagram



Description

- (*constructor*) : only the default constructor will be used. It is called from `Destroyer.run` as first call.
- `beforeDeletion` will be called after execute files was performed. The given list contains all *File* objects which are marked for deletion.
- `isFileListener` determines whether the next two methods are called or not.
- `beforeDelete` is the method which, is called before a single file is deleted. The *File* object is given as parameter.
- `afterDelete` will be invoked after the delete call for a single file.
- `afterDeletion` is the last call before the cleanup of created data is performed.

7.2.2 Package Path

Custom actions must always implement one of the given listener interfaces. As mentioned above, it is also possible to derive from one of the "Simple" listeners. The package path is facultative, only the class name must be unique over all custom actions. The preparation of a custom action for providing it with an installation is very similar to panels. Custom actions must also be packed into a jar file with the name of the custom action class name. This jar file should be placed in [IzPackRoot]/bin/customActions, may be

```
[IzPackRoot]/bin/customActions/MyCompilerListener.jar
[IzPackRoot]/bin/customActions/MyInstallerListener.jar
[IzPackRoot]/bin/customActions/MyUninstallerListener.jar
```

In the default Ant definition file (build.xml) there are some targets for this stuff.

7.2.3 Correlated Stuff

Native Libraries for Uninstallation

If a custom action uses JNI at installation time, often the associated uninstall custom action needs JNI too. For this situation it is possible to declare a native library for uninstallation. The only work to do is to add a **stage** attribute to the **native** tag in the install xml file like

```
<!-- The native section. We specify here our os dependant
libs.--> <native type="3rdparty"
name="MyOSHelper.dll"stage="both" >
  <os family="windows" />
</native>
```

The needed additional classes are packed into lib/uninstaller-ext.jar. If a native library is defined for uninstallation, this file will also be packed into the installer.jar as IzPack.uninstaller-ext and used at its right position.

7.3 What You Have To Do

Follow the steps that are needed to create and use custom actions with the "normal" source environment (not standalone compiler) using Ant. Of course, it works also with the standalone compiler.

7.3.1 Custom Actions at Packaging (CompilerListener)

- Implement `com.izforge.izpack.event.CompilerListener` or extend `com.izforge.izpack.event.SimpleCompilerListener`. Place it as `[IzPackRoot]/src/lib/[MyPackagePath]/MyCompilerListener.java`.
- Add a "compile.simple" antcall in to `[IzPackRoot]/src/build.xml`.

```
<antcall target="compile.listener.simple">
  <param name="listener" value="MyCompilerListener"/>
  <param name="listener-dir" value="MyCompilerListener"/>
  <param name="listener-include" value="[MyPackagePath]"/>
</antcall>
```

- Run `[IzPackRoot]/src/build.xml`.
- Add a "listeners" ELEMENT with a "listener" ELEMENT with a "compiler" attribute in to `[MyProjectPath]/install.xml`

```
<listeners>
  <listener compiler="MyCompilerListener" />
</listeners>
```

- Compile with

```
java -jar [IzPackRoot]/lib/compiler.jar -HOME [IzPackRoot]
[MyProjectPath]/install.xml -b [MyProductPath] -o
[MyBuildPath]/install.jar
```

- Test it

7.3.2 Custom Actions at Installation Time (InstallerListener)

Perform the same steps as described in [7.3.1](#), replace all occurrences of "CompilerListener" with "InstallerListener" and "compiler" with "installer".

7.3.3 Custom Actions at Uninstallation Time (UninstallerListener)

Perform the same steps as described in [7.3.1](#), replace all occurrences of "CompilerListener" with "UninstallerListener" and "compiler" with "uninstaller".

7.4 Example

Let us say, we want to set access rights for files and directories on Unix. The Java sources are placed in the directory

[IzPackRoot]/sample/src/com/myCompany/tools/install/listener. There are the files ChmodCompilerListener.java and ChmodInstallerListener.java.

- Copy the files too [IzPackRoot]/src/lib/com/myCompany/tools/install/listener
- In [IzPackRoot]/src/build.xml there are the lines

```
<!-- CUSTOM ACTION test START  
CUSTOM ACTION test END -->
```

Uncomment them (activate the lines between them).

- Build IzPack new.
- Compile a test installation with

```
java -jar [IzPackRoot]/lib/compiler.jar -HOME [IzPackRoot]  
[IzPackRoot]/sample/listener/install.xml  
-b [IzPackRoot]/sample/listener -o  
[IzPackRoot]/sample/listener/install.jar
```

- Install it

```
java -jar install.jar
```

7.5 Ant Actions (InstallerListener and UninstallerListener)

In this section the common ant task custom actions are described in detail. It is only for developers who are not acquainted with IzPack or its custom actions. In addition to the basics there are some recapitulations of the common custom action techniques and some hints for pitfalls.

In the package `com.izforge.izpack.event` there are the ant related custom actions `AntActionInstallerListener` and `AntActionUninstallerListener`. As recapitulation, to add any custom action a reference in `install.xml` will be needed, as example:


```

<listeners>
  <listener installer="AntActionInstallerListener"
            uninstaller="AntActionUninstallerListener" />
</listeners>

```

For all referenced listeners a jar file with the same name must exist in [IzPackRoot]/bin/customActions. If compilation (packaging) fails with a "not found" error, first verify, that the jar file exists. If not, create it.

With this custom action it is possible to perform ant calls at installation and/or uninstallation time. It is not only a wrapper for a comand-line ant call, but also an intersected description file defining what target of the ant build file should be performed at what time of (un)installation and specifies which properties for what IzPack **pack** are to be used. The intersected description file is written as XML, the corresponding dtd is placed in src/dtd/event/antaction.dtd. The description file should be declared as a resource in the install.xml with the id `AntActionsSpec.xml` e.g.

```

<resources>
  ...
  <res id="AntActionsSpec.xml" src="myInstallSpecs/MyAntActionsSpec.xml" />
  ...
</resources>

```

The precise spelling of the id is important. The base path of `src` is the installation project path. If you want to use ant, you have to specify it here. IzPack is designed for running without dependencies on external software or libraries. Therefore it is necessary to include everything needed, in this case ant self. The field `<jar>` in installation.xml is predestinated for such cases, e.g.

```

<jar src="jar/ant/ant.jar" stage="both" />

```

Be aware, that an "extended" ant use needs more than one jar, for example often `xercesImpl.jar`. If an obscure "class not found" exception is raised during testing, check first for missing jar files.

For supporting uninstallation the jar field was extended by the attribute **stage**. If an ant uninstaller custom action is used, the uninstaller also needs the jar files. If **stage** is "both" or "uninstall", the contents of the referenced jar file will be packed into `uninstaller.jar`. Be aware that not the jar file itself, but the contents of it are required. This implies, that the paths of the contained files are unique and the information in `meta-inf/Manifest.mf` will be lost.

7.5.1 The Basic XML Struture

An ant action will be defined in the resource with the id "AntActionsSpec.xml". Sometimes it will help to lock into [IzPackRoot]/src/dtd/event/antaction.dtd or validate a written xml file with the dtd.

On this xml file a substitution will be performed using all defined **IzPack** variables. It is performed just before processing the packs. This is a common way of loading spec files into custom actions. For more information see method `com.izforge.izpack.util.SpecHelper.readSpec`. If you want to substitute some custom item, simply add a variable via `idata.setVariable` in a custom panel before `InstallPanel`. The given variable name (id) should be written into the xml file in the common variable notation.

The top level XML section is called `<antactions>`. Only one is possible. The `<antactions>` are segregated in one or more `<pack>` elements. The single attribute `<name>` of the `<pack>` corresponds to the same structure in `install.xml` (for more information see also `installation.dtd`). Only the "things" included in the `<pack>` are performed, if a pack with the same name was chosen to be installed. The "things" to be done to self are defined by the element `<antcall>` (without ssss).

The `<antcall>` takes the following attributes:

- **order**: required. Determine at what point of installation the antcalls defined by element `target` should be performed. Possible are `beforepack`, `afterpack`, `beforepacks` or `afterpacks`. Be aware that with `beforepack(s)` there are no installed files and also no installed build file. With this order only preexistent build files are useable.
- **uninstall_order**: optional. Determine at what point of uninstallation the antcalls defined by element `uninstall_target` should be performed. Possible are `beforedeletion` and `afterdeletion`. As opposed to the behaviour of `order` the referenced files are also accessible in the order `afterdeletion`. The uninstaller action copies the files into tempfiles before deletion which are marked as `deleteOnExit`.
- **quiet**: optional. To quit or not. Possible are yes or no. Default is no.
- **verbose**: optional. To output verbose information or not. Possible are yes or no. Default is no.
- **logfile**: optional. Path of the file for logging should be performed. The logfile should be not marked for uninstallation otherwise it will be deleted too.

- **buildfile:** required. Path of the file which contains the antcall. This is the file you normally use as `-buildfile` during an ant call via the command line. In this file variables are not substituted. For substitution there are properties in ant which can be used. Never write an IzPack variable in an ant buildfile.
- **messageid:** optional. A string ID which refers to `bin/langpacks/installer/<lang>.xml`. If it is defined, the message will be displayed in the InstallPanel whilst performing the ant call.

In addition to the possible attributes there are some elements. All elements can be defined more than one time in one `<antcall>`. All are optional, but with no `<target>` element the `<antcall>` makes no sense. Do not confuse the following: "required"s are related to the attributes of the elements, not to the elements themselves.

`<property>`: define a property

Property to be used with all `targets` and `uninstall_targets` which are defined for this antcall.

- **name:** required. The name (id) of the property.
- **value:** required. The value of the property.

`<propertyfile>`: define properties in a file

Properties to be used with all `targets` and `uninstall_targets` which are defined for this antcall given by the path of a properties file.

- **path:** required. Path of a file which contains properties in the syntax which is used by ant. Some ant calls need properties files. For these this element is used. One way to fill specific data into it is to create a new file in a custom panel and fill it with values given by input fields. The file path can be set at installation time, if there is a variable in `AntActionSpec.xml` and an IzPack variable was defined before InstallPanel. That file can be only created with `deleteOnExit`, if no `<uninstall_target>` was defined in this `<antcall>`. This implies, that other `<antcall>`s can have a `<uninstall_target>`.

<target>: target to call at installation

Targets to perform with this antcall at installation time. The targets should be defined in the given buildfile or else an ant exception will be raised. This is that what you use, if you don't want to perform the default target. e.g. cleaning the IzPack project with `ant clean`

- **name**: required. The name of the target.

<uninstall_target>: target to call on uninstallation

Targets to perform with this antcall at uninstallation time. The targets should be defined in the given buildfile otherwise an ant exception will be raised. With this target it will be possible to undo the things done at installation time.

- **name**: required. The name of the uninstall target.

Appendix A

The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you

conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program

with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER

PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>
```

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B

The Commons Creative Attribution-NonCommercial- ShareAlike License

A friendly summary of the license terms is available at <http://creativecommons.org/licenses/by-nc-sa/1.0/>. The following are the full legal terms which govern this documentation.

Creative Commons

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 1.0

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS DRAFT LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF

SUCH TERMS AND CONDITIONS.

1. Definitions

- a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
- b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License.
- c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
- d. "Original Author" means the individual or entity who created the Work.
- e. "Work" means the copyrightable work of authorship offered under the terms of this License.
- f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
- b. to create and reproduce Derivative Works;
- c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
- d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio

transmission Derivative Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.
- b. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work

apart from the Derivative Work itself to be made subject to the terms of this License.

- c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- d. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

- a. By offering the Work for public release under this License, Licensors represents and warrants that, to the best of Licensors' knowledge after reasonable inquiry:
 - i. Licensors has secured all rights in the Work necessary to grant the license rights hereunder and to permit the lawful exercise of the rights granted hereunder without You having any obligation to pay any royalties, compulsory license fees, residuals or any other payments;
 - ii. The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.
- b. EXCEPT AS EXPRESSLY STATED IN THIS LICENSE OR OTHERWISE AGREED IN WRITING OR REQUIRED BY APPLICABLE LAW, THE WORK IS LICENSED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES REGARDING THE CONTENTS OR ACCURACY OF THE WORK.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, AND EXCEPT FOR DAMAGES ARISING FROM LIABILITY TO A THIRD PARTY RESULTING FROM BREACH OF THE WARRANTIES IN SECTION 5, IN

NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at <http://creativecommons.org/>.